Faster Fourier Transforms via Automatic Program Specialization

Julia Lawall

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Faster Fourier Transforms via Automatic Program Specialization

Julia L. Lawall

N° 3437

Juin 1998

THÈME 2

ISSN 0249-6399

RENNES

Faster Fourier Transforms via Automatic Program Specialization

Julia L. Lawall

Thème 2 — Génie logiciel
et calcul symbolique
Projet Compose

Rapport de recherche n3437 — Juin 1998 — 28 pages

Abstract: Because of its wide applicability, many efficient implementations of the Fast Fourier Transform have been developed. In this paper we propose that efficient implementations can be produced automatically and reliably by partial evaluation. Partial evaluation of an unoptimized implementation produces a speedup of over 7 times. The automatically generated result of partial evaluation has performance comparable to or exceeding that produced by a variety of hand optimizations. We analyze the benefits of partial evaluation at both compile time and run time, and survey related hand-optimization techniques.

Key-words: Partial evaluation, compilation, compile-time optimization, run-time optimization, Fast Fourier Transform

(Résumé : tsvp)

Author's current address: Computer Science Program, Oberlin College, Oberlin, Ohio, 44074, USA, jll@cs.oberlin.edu

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Génération de transformées de Fourier optimisées par spécialisation automatique de programmes

Résumé : De part son domaine d'application très large, la transformée de Fourier rapide (FFT) a fait l'objet de nombreuses études visant à optimiser son implémentation. Dans cet article, nous proposons d'utiliser la technique d'évaluation partielle, et plus précisément le spécialiseur Tempo, pour générer de façon automatique et fiable une mise en œuvre efficace de la FFT. En spécialisant une implémentation naïve de la FFT, nous obtenons un facteur d'accélération allant jusqu'à 7. De plus, cette FFT optimisée automatiquement est au moins aussi efficace qu'une implémentation optimisée au moyen de techniques manuelles. Une caractéristique innovante de Tempo est de permettre la spécialisation soit lors de la compilation ou à l'exécution. Nous analysons l'utilisation de ces deux stratégies sur la FFT, et décrivons leurs domaines d'application respectifs.

Mots-clé : évaluation partielle, compilation, optimisation à la compilation, optimisation à l'execution, transformée de Fourier rapide (FFT)

1 Introduction

The Fourier transform and its inverse are widely used in a variety of scientific applications, such as audio and image processing [32], integration [32], and calculation using very large numbers [3, 33]. The transform converts a function defined in terms of time to a function defined in terms of frequency. When a function is defined over the frequency domain, some expensive calculations are more tractable. This technique was made practical by the development of the Fast Fourier Transform (FFT) [11], which uses a divide-and-conquer algorithm to calculate the Fourier transform of a function represented as a discrete set of evenly-spaced data points. The divide-and-conquer algorithm reduces the complexity from $O(n^2)$ to $O(n \log n)$, where n is the number of data points.

Despite the significantly improved performance, the FFT remains an expensive operation. Many computations spend a substantial amount of time performing FFT's. For example, the `125.turb3d` benchmark of the SPEC95 Benchmark suite [12] spends about 40% of the time performing FFT's of 32 or 64 elements, using a hand-optimized implementation. Much effort has gone into hand-optimizing implementations of the algorithm. These optimizations include using recurrences to limit the number of calls to sine and cosine, eliminating the calls to these math library routines completely by reimplementing them more efficiently or using tables, reducing the number of real multiplications, and unrolling loops.

Hand optimization can be tedious and error-prone. Partial evaluation is an automatic program transformation that specializes a module with respect to a subset of its input. Partial evaluation performs aggressive, interprocedural constant propagation based on the known input. Improvements that can be obtained include constant folding, loop unrolling, strength reduction, and simplification of control flow. In contrast to hand optimization, optimization by partial evaluation is automatic and reliable. Correctness follows from the correctness of the partial evaluator.

In this paper we investigate whether partial evaluation is a suitable tool for generating an efficient FFT implementation. We address two aspects of this question. First, we determine what are the best results we can obtain automatically by partial evaluation of an unoptimized implementation that directly follows the mathematical algorithm. To obtain the best results from partial evaluation, we have to modify the implementation slightly. We find that partial evaluation improves the unoptimized implementation over 7 times when the input contains 16 elements and over 3 times when the input contains 512 elements. Second, we compare the best results that can be obtained with partial evaluation to the performance of other optimized implementations. We find that partial evaluation matches or substantially exceeds the benefits of other implementation techniques.

The rest of this paper is organized as follows: Section 2 presents an overview of partial evaluation. Section 3 assesses the opportunities for specialization in the FFT algorithm, and estimates the speedup that can be obtained. Section 4 carries out the specialization of a simple implementation of the FFT. In Sections 5 and 6 we slightly rewrite the source program to get better results from compile-time and run-time specialization, respectively. Next, Section 7 describes optimization techniques that have been applied by hand in FFT implementations. The implementations considered are all publicly available. We compare the performance of the specialized code of the simple implementation with that of the hand-optimized implementations, and with the results of specializing the latter. Finally, Section 8 describes other related work and 9 concludes.

2 Overview of partial evaluation

Partial evaluation is an automatic program transformation that specializes a module with respect to part of its input. Values that depend only on the known input and on program constants are said to be *static*. Other values are said to be *dynamic*. An expression that can be simplified based on only the static data is said to be *reducible*. Other expressions, which must be copied into the specialized program, are said to be *residualizable*. We consider only *offline* partial evaluation. Offline partial evaluation consists of two phases. The first[1] is binding-time analysis, which annotates each expression as either reducible or residualizable. The second is specialization, which simulates evaluation following the annotations of the binding-time analysis. A thorough introduction to partial evaluation (in the functional setting) is found in Jones *et al.*'s textbook [22].

[1] In practice, many preprocessing analyses may be involved, such as alias analysis, use-def analysis, etc. [9]. For the purposes of this paper, we refer to all of these as binding-time analysis.