Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Automatically deriving cost models for structured parallel processes using hylomorphisms

David Castro *, Kevin Hammond, Susmit Sarkar, Yasir Alguwaifli

School of Computer Science, University of St Andrews, St Andrews, UK

HIGHLIGHTS

- An operational semantics of a queue language for streaming computations.
- An operational semantics of algorithmic skeletons using this queue language.
- Derive cost equations for algorithmic skeletons from the operational semantics.
- Cost equations for algorithmic skeletons are combined with sized types.

ARTICLE INFO

Article history:
Received 24 March 2017
Accepted 21 April 2017
Available online xxxx

Keywords:
Operational semantics
Algorithmic skeletons
Cost models
Hylomorphisms

ABSTRACT

Structured parallelism using nested algorithmic skeletons can greatly ease the task of writing parallel software, since common, but hard-to-debug, problems such as *race conditions* are eliminated *by design*. However, choosing the best combination of algorithmic skeletons to yield good parallel speedups for a specific program on a specific parallel architecture is still a difficult problem. This paper uses the unifying notion of *hylomorphisms*, a general recursion pattern, to make it possible to reason about both the functional correctness properties and the extra-functional timing properties of structured parallel programs. We have previously used hylomorphisms to provide a denotational semantics for skeletons, and proved that a given parallel structure for a program satisfies functional correctness. This paper expands on this theme, providing a simple operational semantics for algorithmic skeletons and a cost semantics that can be automatically derived from that operational semantics. We prove that both semantics are sound with respect to our previously defined denotational semantics. This means that we can now automatically and statically choose a provably optimal parallel structure for a given program with respect to a cost model for a (class of) parallel architecture. By deriving an *automatic amortised analysis* from our cost model, we can also accurately predict parallel runtimes and speedups.

1. Introduction

In previous work [1], we have defined a type-based mechanism for reasoning about the safe introduction of parallelism using a structured parallel approach. Our approach allows us to extract parallel program structure as a type. Given a suitable model of a program's execution cost (e.g. in terms of its time performance), we can reason formally about performance improvements for alternative parallelisations, and so select a *provably optimal* parallel implementation. In this paper, we show how to derive appropriate cost models formally from the parallel structure of a program, using a new queue-based operational semantics (Fig. 1). This gives a completely formal system for reasoning about the performance of structured parallel programs. We combine related work on *algorithmic skeletons* and *recursion schemes*. Algorithmic skeletons [2] are parametric implementations of common patterns of parallel programming. Using a pattern/skeleton approach, the programmer can design and implement a parallel program in a top-down manner. For example, the programmer could first identify the parallel patterns that occur in a particular piece of software, then select the patterns that potentially lead to the best speedups, and finally select a suitable implementation for those patterns, as a composition of one or more algorithmic skeletons. This composition then exposes the *parallel structure* of the implementation. Developing an equational theory for easily, and automatically, changing the

* Corresponding author.
E-mail addresses: dc84@st-andrews.ac.uk (D. Castro), kevin@kevinhammond.net (K. Hammond), ss265@st-andrews.ac.uk (S. Sarkar), ya8@st-andrews.ac.uk (Y. Alguwaifli).

http://dx.doi.org/10.1016/j.future.2017.04.035
0167-739X/

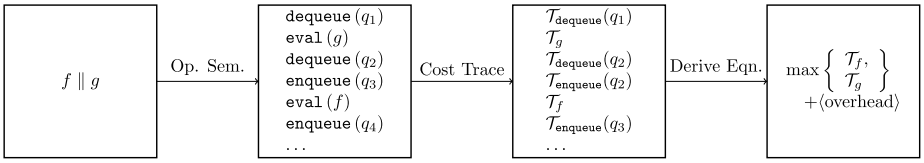

Fig. 1. Deriving cost equations from operational semantics.

parallel structure of a program has been the subject of research in the skeletons community for the last two decades [3–6]. Over a similar timeframe, in the functional programming community, a structured form of recursion has been explored, in the form of *patterns of recursion*, or *recursion schemes* [7]. Research on this topic has brought many improvements to equational reasoning in functional languages. One example is the application of the laws and properties of recursion schemes to the well-known *deforestation* optimisation [8,9].

There is an obvious connection between algorithmic skeletons and recursion schemes: algorithmic skeletons are essentially *higher order functions* that implement some common pattern of parallelism. This has already been exploited a number of times [3,10–12]. In [1], we expand on this connection, using the fact that a large number of recursion patterns can be represented as instances of a more general pattern, a *hylomorphism*. The basic idea is to provide a single unifying framework for reasoning both about program transformations and about parallel execution times. This unifying framework provides a type-level abstraction of the *program structure*, given as a combination of the *hylomorphisms* and *algorithmic skeletons* that are used to implement a particular program. This allows us to define a type-based mechanism for reasoning about the safe introduction of parallelism: specific combinations of hylomorphisms can be "replaced" by specific combinations of algorithmic skeletons. We provide strong static guarantees that the resulting program will be functionally equivalent to the original one. Given a suitable *model* for the cost of a parallel implementation (e.g. in terms of execution time), we can then use this type-based approach to reason formally about performance improvements for alternative parallelisations, and so to select a *provably optimal* parallel implementation.

1.1. Novel contributions

In this paper, we show how to formally derive appropriate cost models from the parallel structure of a program, using a new queue-based operational semantics. This gives a completely formal system for reasoning about the performance of structured parallel programs. The main novel contributions of the paper are:

- We define the operational semantics of a *queue language* that is powerful enough to describe the operational semantics of a number of algorithmic skeletons, and small and restrictive enough to facilitate reasoning about correctness and execution times (Section 4).
- We define the *operational semantics* of a number of key algorithmic skeletons using this queue language (Section 5).
- We derive a set of *cost equations* for a number of algorithmic skeletons from the operational semantics in a systematic way. We combine these cost equations with the notion of *sized types*, and sketch how this process can be automated (Section 6).

1.2. Motivating example

We illustrate our approach using the *image merge* example from [1]. The purpose of the `imgMerge` function is to `mark` and then `merge` pairs of images.

$$\texttt{imgMerge} \;:\; \mathsf{List}(\texttt{Img} \times \texttt{Img}) \rightarrow \mathsf{List}\ \texttt{Img}$$
$$\texttt{imgMerge} \;=\; \mathsf{map}\ (\texttt{merge} \circ \texttt{mark})$$

This has many possible, semantically equivalent, parallel implementations.

$$\texttt{imgMerge}_1 \;=\; \mathsf{farm}\ n\ (\mathsf{fun}\ (\texttt{merge} \circ \texttt{mark}))$$
$$\texttt{imgMerge}_2 \;=\; \mathsf{farm}\ m\ (\mathsf{fun}\ \texttt{mark}) \parallel \mathsf{farm}\ n\ (\mathsf{fun}\ \texttt{merge})$$
$$\texttt{imgMerge}_3 \;=\; \mathsf{farm}\ n\ (\mathsf{fun}\ \texttt{mark}) \parallel \mathsf{fun}\ \texttt{merge}$$
$$\ldots$$

here, farm n is a *task farm* skeleton that replicates its argument n times, fun captures primitive (sequential) functions, $\circ$ is the normal function composition, and $\parallel$ is a parallel pipeline, i.e. the parallel composition of two skeletons. The structure of each parallel implementation can be lifted into an appropriate type signature, so that

$$\texttt{imgMerge}_1 \;:\; \mathsf{List}(\texttt{Img} \times \texttt{Img}) \xrightarrow{\mathrm{FARM}_n(\mathrm{FUN}\ (\texttt{merge}\circ\texttt{mark}))} \mathsf{List}\ \texttt{Img}$$

By combining well-known properties of algorithmic skeletons [13] and *hylomorphisms* [7], we can define a *convertibility* relation that relates each of these types for *imgMerge* to any of the other types. Moreover, this same relation also allows the underlying program to be automatically rewritten so that it matches the new type. We can thus automatically select any valid parallel implementation of *imgMerge* simply by changing the type, and without changing its definition. Finally, if we have a cost model for these types, we can now reason about costs, and automatically select the provably best parallel implementation for *imgMerge*, or, indeed, for any appropriately structured parallel program.

2. Algorithmic skeletons

We use *algorithmic skeletons* [2] to represent structured parallel patterns. Skeletons are parameterised templates (*higher-order functions*) that capture the structure of a parallel program. That is, they implement a specific parallel *pattern*, that can be instantiated to produce a specific parallel *algorithm*. In this paper, as in our earlier work [1,14], we use a "pluggable" approach, where all skeletons are *streaming* entities (that is, they operate over multiple inputs and produce multiple results). Each skeleton takes its inputs from an *input queue*, and produces a *result queue*. This approach allows skeletons to be easily nested or linked together into more complex structures, whose sub-components are connected *via* intermediate queues. In this paper, we will consider four common parallel skeletons, *task farms, parallel pipelines, feedbacks*, and *parallel divide-and-conquer*, plus one basic building block, *structured functions*.

Please cite this article in press as: D. Castro, et al., Automatically deriving cost models for structured parallel processes using hylomorphisms, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.04.035.