Chapter 8
Conclusions

This book has demonstrated the effectiveness of a Threaded Interpretive Graph Reduction Engine for executing combinator graph reduction. The performance of TIGRE has been analyzed with respect to changes in computer architecture. While special-purpose hardware can give measurable performance improvements, the improvements are not significant enough to merit building a custom CPU for the execution of TIGRE programs. However, the exercise of creating the conceptual design for a fast TIGRE processor shows that several architectural features including cache management strategy, hardware stack buffering, and support of fast subroutine calls can significantly improve performance if available on stock hardware.

8.1. CONTRIBUTIONS OF THIS RESEARCH

A summary of the contributions of research reported here is as follows:

- An execution model for combinator graph reduction has been designed. This model is based on treating the combinator graphs as executable threaded program graphs, instead of data graph structures. Although the use of threaded execution for graph reduction has been previously reported, this is the first comprehensive treatment presenting discussion of implementation techniques, performance, and architectural considerations.
- TIGRE, the Threaded Interpretive Graph Reduction Engine, has been designed and implemented on a number of platforms. TIGRE has been shown to be a fast graph reducer, based on published performance data for other combinator reduction schemes.
- TIGRE is able to take advantage of supercombinator compilation and strictness analysis to improve efficiency over the Turner Set of combinators. These advanced compilation tech-

niques do not require modifications to the TIGRE execution scheme, showing the generality of the TIGRE approach.

- TIGRE is shown to be able to perform well within an order of magnitude of performance of imperative languages and strict functional programming languages, for small programs. Thus lazy functional programming and other programming methodologies built on pure combinator graph reduction are viable for research and, with further improvements, for general programming use.
- Cache simulations for TIGRE show that combinator graph reduction has some unexpected cache behavior. Specifically, programs show very high spatial locality. Additionally, the write-allocate characteristic of cache memories, which is given little attention in most architectural studies, proves to be crucial to attaining good hit ratios for graph reduction and, indeed, for any program using a compacting garbage collector.
- Other architectural analysis for TIGRE is given, listing desirable architectural features for combinator graph reduction. These features are desirable to the extent that they can be found on existing architectures, but do not offer the order-of-magnitude speedups required to justify building specialized processors.

8.2. AREAS FOR FURTHER RESEARCH

The development of TIGRE has yielded new insights and new questions at every step. Many research areas are now ready for exploration. Some of the more important areas are listed below.

- **Parallel graph reduction**. A problem with parallel graph reduction in the past has been one of practicality. If individual graph reduction nodes could not be made to go within a factor of 100 as fast as a uniprocessor running an imperative language, there seemed little point in building a 100 node processor which could not possibly go as fast as single workstation. TIGRE thus makes it feasible to design a parallel graph reduction engine that can potentially run programs more quickly than a uniprocessor using imperative languages. The Grip project (Peyton Jones *et al.* 1987) has used similar reasoning to justify a parallel closure reduction machine based on TIM, but TIGRE makes parallel pure graph reduction viable. Graph reduction is a simpler, more obvious program manipulation technique than other methods, and therefore *may* allow better insight into parallel execution issues.

- **Further software tool development**. The work reported here has focused on the core of TIGRE and, in particular, the actual graph reduction engine. The results obtained form a favorable feasibility analysis for the TIGRE approach to graph reduction. However, many more software tools will be needed before TIGRE can come into general use by the research community. In particular, adaptation of a front-end for some functional programming language, completion of a fully automatic supercombinator compiler and a creation of a usable interactive shell are required to make TIGRE useful for software development.
- **Further architectural measurements**. The initial measurements of TIGRE behavior are limited by a lack of extensive software support. Additionally, large lazy functional programs are difficult to find and, even when available, impossible to run without good compiler support. Therefore, it would be prudent to repeat some of the experiments on TIGRE when a larger software base is available. Also, it would be revealing to compare TIGRE against other abstract machines using a comprehensive benchmark suite with comparable implementations and compilers. This would not only improve understanding of the strengths and weaknesses of TIGRE, but also of common architectural requirements for combinator reduction techniques in general.
- **Formal analysis of TIGRE**. A formal mathematical representation of TIGRE might permit the analysis of TIGRE optimizations and a characterization of TIGRE with respect to other approaches. It could help discover the fundamental differences between TIGRE and other reducers such as TIM and the G-Machine.