Research Proposal: Register Allocation for Irregular Architectures

David Koes

January 14, 2004

1 Problem

Register allocation is one of the most important optimizations a compiler performs and is becoming all the more important as the gap between processor speeds and memory access times widens. Correspondingly, a lot of effort has been spent attempting to improve traditional allocators. However, most of these attempts dealt with regular, RISC-like architectures. Published attempts to make algorithms applicable to architectures that are irregular in their use of registers have yielded several incompatible extensions that handle only a small subset of the irregularities seen in modern architectures (or so say Smith and Holloway [12]).

Irregular architectures, epitomized by IA32, may have such features as register pairing (where a value must be stored in two adjacent registers), subword registers (where a value is stored and accessed in only part of a register), and register classes (where instructions can only use a subset of the total registers). In addition to the ubiquitous x86 architecture, features of irregular architectures are wide spread in embedded architectures. The Starcore, MCore, 68k, and ColdFire processors have address/data register banks and instructions are restricted in what class of register they may use. The x86 also restricts what sort of registers specific instructions can use, but not so cleanly. PA-RISC and SPARC processors use register pairing to create double valued floating point registers and most software implementations of 64 bit long longs use paired integer registers. The Broadcom Firepath, I think, implements 64 bit integers in hardware using paired integer registers. Many embedded processors, especially those with DSP connections, provide support for subword register accesses.

The problem I would (very much so) like to work on is developing new algorithms that are demonstrably better than existing approaches for targeting irregular architectures in an embedded space. This means the algorithms would fully exploit the irregularities of the architecture (rather than just cope with them) and, in addition to optimizing for speed, optimize for code size. Because of the irregularities of register usage (in particular, the prevalence of addressing modes in CISC architectures), minimizing the number of spills is not equivalent to minimizing code size (a distinction I've yet to see addressed in the literature).

2 Some Related Work

Brigg's thesis [2] discusses the problem of register pairing and solves it by modifying the interference graph appropriately. Smith and Holloway [12] directly address the challenge of compiling for irregular architectures, in particular register classes and multi-register sets, by modifying the heuristics guiding the graph coloring algorithm so it deals with both register pairs and different register classes. They fail to demonstrate that their approach is better than other approaches. Tallam and Gupta [13] exploit the use of subword data by applications and the existence of subword referencing instructions in embedded systems. They are successful in reducing the register requirements of some functions. Their work (which is pretty impressive, I think) could be extended to be more aware of the trade-offs involved in packing subword data. Furthermore, they seem to assume subword access abilities that current architectures do not posses (or at least not efficiently). The work of Kolson et al [6] and Scholz [11] are the closest to what I want to do. Kolson combines loop unrolling with register allocation. While his approach can cope with irregular architectures, it does not seem to take advantage of them. Scholz feels (and I agree) that compiling for irregular architectures is not well served by graph coloring and instead reduces the problem to the partitioned boolean quadratic optimization

problem (which is NP-complete) and then solves in supposedly linear time using heuristic solvers. It would be necessary to demonstrate superiority over these approaches.

Provably optimal register allocation has been achieved using integer linear programming[7] however this technique is unreasonably slow. Methods have been proposed that improve the running time of ILP [3] [1]. While these improve the speed of the ILP pass they are either still too slow or involve other trade offs reducing the optimality. More tractable optimal (or near-optimal) solutions can be found for the local register allocation problem [8] [9] [4].

Optimal register allocation usually refers to the allocation of registers that results in the fewest spills (multiplied by a spill cost that increases with loop nests). However, in the embedded space other factors, such as code size and power consumption, may be more important than speed [5] [10].

3 Goals

The goal is to come up with demonstrably better register allocation techniques for irregular architectures in the embedded systems domain. This goal has several components:

- **demonstrably better** Ideally this means something is provably optimal or near so. Alternatively empirical comparisons to the current state of the art are always acceptable.
- **irregular architectures** The allocator should exploit the features of irregular architectures, not just work around them. For example, spilling to different register classes instead of memory, packing subword data into single registers, and spilling paired registers to unpaired registers instead of memory.
- **embedded systems domain** In embedded applications reducing the number of spills is not always what is desired. Reducing code size may be more important. Because of the irregularity of the architecture this is particularly challenging (CISC memory operands make some spills cheap and others expensive).

There are also several more practical goals:

- Have a paper ready for the February 6 deadline for LCTES (ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems).
- Once the paper is done, create a talk to fulfill the speaking requirement that includes future research directions.
- Work in gcc since it targets many embedded architectures allowing for multiple evaluations. If it's any good, submit the code back.

4 Approach

I'd like to start by pursuing my goal with respect to local register allocation only. This is generally considered to be an easier problem. People have actually been able to come up with useful proofs with respect to local register allocation. Furthermore, local register allocation is an important part of many conventional compilers. New algorithms for irregular architectures would have a direct impact, even if limited to local register allocation.

Once we know how to do local allocation properly, the hope would be that this insight could be used to implement an improved global allocator. I think Scholz [11] has the right idea that graph coloring is *not* the appropriate metaphor for irregular register allocation.

5 Timeline

- **Week of Jan 12** Complete literature survey. Write up related work.

- **Week of Jan 19** Extend existing local register allocation algorithms to be optimal with respect to program size on a CISC architecture (spill costs not fixed, but depend upon use of variable). Implement within gcc.
- **Week of Jan 26** Extend algorithm to exploit multiple register classes. Write up algorithm and get preliminary results.
- **Week of Feb 2** If time permits, extend algorithm to perform subword packing. I doubt there will be time to actually write a bitvalue pass. Instead it will be necessary to use the natural size of the variable (but that's okay, since it's the algorithm that matters, not the analysis which provides the data to the algorithm). Write the paper.
- **Week of Feb 9** Put together talk on work. Schedule speaking requirement talk.

References

[1] Andrew W. Appel and Lal George. Optimal spilling for cisc machines with few registers. In *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, pages 243–253. ACM Press, 2001.

[2] Preston Briggs. Register allocation via graph coloring. Technical Report CRPC-TR92218, Rice University, Houston, TX, April 1992.

[3] Changqing Fu and Kent Wilken. A faster optimal register allocator. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 245–256. IEEE Computer Society Press, 2002.

[4] Wei-Chung Hsu, Charles N. Fisher, and James R. Goodman. On the minimization of loads/stores in local register allocation. *IEEE Trans. Softw. Eng.*, 15(10):1252–1260, 1989.

[5] Mike Jochen and Amie Souter. Program analysis and optimization for embedded systems.

[6] David J. Kolson, Alexandru Nicolau, Nikil Dutt, and Ken Kennedy. Optimal register assignment to loops for embedded code generation. *ACM Transactions on Design Automation of Electronic Systems.*, 1(2):251–279, 1996.

[7] D. Kstner and M. Langenbach. Integer linear programming vs. graph-based methods in code generation.

[8] Vincenzo Liberatore, Martin Farach, and Ulrich Kremer. Hardness and algorithms for local register allocation.

[9] Waleed M. Meleis and Edward S. Davidson. Optimal local register allocation for a multiple-issue machine. In *Proceedings of the 8th international conference on Supercomputing*, pages 107–116. ACM Press, 1994.

[10] Mayur Naik and Jens Palsberg. Compiling with code-size constraints. In *Proceedings of the joint conference on Languages, compilers and tools for embedded systems*, pages 120–129. ACM Press, 2002.

[11] Bernhard Scholz and Erik Eckstein. Register allocation for irregular architectures. In *Proceedings of the joint conference on Languages, compilers and tools for embedded systems*, pages 139–148. ACM Press, 2002.

[12] Michael D. Smith and Glenn Holloway. Graph-coloring register allocation for irregular architectures.

[13] S. Tallam and R. Gupta. Bitwidth aware global register allocation, 2002.