An Integrated Technique for Instruction Scheduling and Register Allocation Based on Subgraph Isomorphism

Lucas Silva, Richard Silva, and Ricardo Santos *

High Performance Computing Systems Laboratory
School of Computing
Federal University of Mato Grosso do Sul
Campo Grande - MS - Brazil
{lucascsilva,richardsteffano}@gmail.com,ricardo@facom.ufms.br
http://lscad.facom.ufms.br

Abstract. This work aims at providing an integrated Instruction Scheduling and Register Allocation algorithm based on Subgraph Isomorphism theory. The proposed algorithm is based on the modelling of the hardware resources of the target processor architecture as a base graph, and the representation of the input program as DAGs with vertices representing program instructions, input, and output operands, and edges representing the dependence among instructions. The inputs for the algorithm are Directed Acyclic Graphs (DAGs) G_1 of the program and a base graph G_2 that represents the target architecture. The output is a graph G_2' (isomorphic to G_1) representing the results of the scheduling and register allocation.

Keywords: Instruction Scheduling, Register Allocation, LLVM

1 Introduction

It is well known that instruction scheduling and register allocation are important phases in code generation. There are many proposals [3,4,5,6] focusing on integrated approaches for instruction scheduling and register allocation. There are approaches using from integer linear programming models up to multi-commodity network flow models. Those techniques propose alternatives to solve both phases together at the expense of poor compiler performance.

This work extends a previous proposal [2] by combining together instruction scheduling and register allocation and solving them using the Subgraph isomorphism theory on the LLVM compiler [1]. The key aspect of our technique is the modelling of the hardware resources (functional units, register files and their interconnections) as a base graph, and the representation of the input

* The authors thank Brazilian Research Agencies CAPES, CNPq, and Fundect for their financial support to this work and related research at the High Performance Computing Systems Laboratory (LSCAD/FACOM/UFMS).

program as DAGs with vertices representing program instructions, input and output operands, and edges representing the dependence among instructions. The inputs for the proposed integrated algorithm are comprised of a number of Directed Acyclic Graphs (DAG) G_1 of the program and a base graph G_2 that represents the target architecture. The output is a graph G_2' isomorphic to G_1 indicating the resources used according to the scheduling and register allocation just performed. Our technique has been used as a viable alternative to generate code for architectures ranging from scalar, and superscalar [2] processor execution models. To demonstrate the feasibility and flexibility of the technique, we have designed our integrated approach for the MIPS target architecture on the LLVM compiler [1].

This paper is organized as follows: Section 2 presents our proposal for an instruction scheduling and register allocation integrated algorithm. The experiments and results are presented in Section 3. The final remarks and proposals for future work are described in Section 4.

2 Instruction Scheduling and Register Allocation Based on Subgraph Isomorphism

Figure 1 sketches a base graph representing the MIPS architecture [8]. For the sake of simplicity, only four registers are presented. Vertices R1, R2, R3, and R4 (circles) represent physical registers and vertices FU_1 and FU_2 (squares) represent functional units (for ALU and MUL/DIV operations).

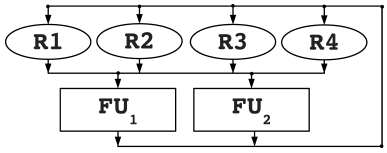
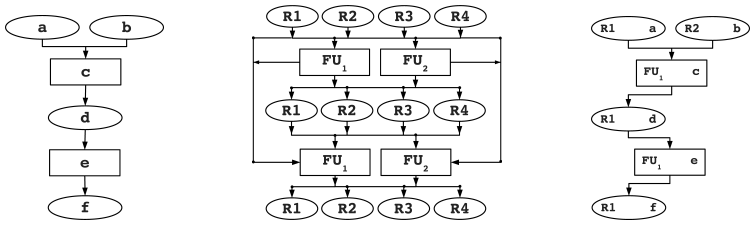


Fig. 1. MIPS graph represented with four registers.

Figure 2 sketches an example of subgraph isomorphism on a base graph representing the MIPS architecture [8]. Figure 2(a) is an input DAG where vertices a, b, d, and f (circles) represent operands inputs/outputs; vertices c and e (squares) represent operations of instructions. Figure 2(b) shows the base graph of the MIPS target processor unrolled to support the subgraph matching to the input DAG. The result of the subgraph isomorphism procedure between graphs in Figures 2(a) and 2(b) is the graph in Figure 2(c).

2.1 An Integrated Algorithm for Scheduling and Register Allocation on the LLVM Compiler

We have carried out a set of heuristics and used the main ideas from the VF subgraph isomorphism library [7] to design and implement our algorithm for



(a) Input Graph (G_1) (b) MIPS Base Graph (G_2) (c) Resulting Graph (G'_2)

Fig. 2. Example of Subgraph Isomorphism on the MIPS base graph.

scheduling and register allocation. The VF algorithm finds an isomorphism if there exist one and can determine the best isomorphism (optimal result).

The implementation of the algorithm has been carried out as a new pass on the LLVM back-end. The first step of the algorithm to determine DAGs sizes (height and width) for DAGs on each basic block. The sum of the heights and widths of the DAGs are passed to a procedure to build up the base graph. Each basic block calls the LLVM scheduler that builds up a graph of Schedule Units. Then, the Schedule Units graph is passed to a topological ordering procedure. The base graph is built using LLVM Machine function classes and unrolled according to the procedure to determine DAGs sizes. Both graphs (Schedule Units graph and Unrolled Base Graph) are passed to the subgraph matching engine. The result of the matching is another graph that is reordered and passed to the emit schedule procedure.

```
Inputs: a DAG G1 from a basic block, a base graph G2
Outputs: a subgraph G21' isomorphic to the input
1) Calculate the unrolling for the base graph G2
2) Create the unrolled base graph G2
3) Perform subgraph matching between G1 and G2
4) If the matching is not possible:
   4.1) if G2 is not large enough:
        4.1.1) increase the unrolling factor
        4.1.2) go to step 2
   4.2) if there is a spill:
        4.2.1) include vertices representing store and load from memory
        4.2.2) update DAG G1 with vertices from 4.2.1
        4.2.3) go to step 3
   4.3) if a matching is not found:
        4.3.1) split up the DAG under matching
        4.3.2) go to step 3
 5) Emit scheduled and register allocated instructions for a basic block
```

Algorithm 1: Steps of the integrated algorithm for scheduling and register allocation based on subgraph isomorphism.

Algorithm 1 describes the main steps of our algorithm. It starts by receiving a DAG from a basic block to the base graph size procedure (step 1) that calculates the unrolling factor for the base graph. The unrolling factor is passed to the base graph creation procedure (step 2). Step 3 performs the subgraph matching between G1 and G2. Currently, the matching is performed by the VF library that runs an exact subgraph matching algorithm (time complexity of $O(V!V)$, $V = max(V_1, V_2)$) with a timeout limit. A matching may not be found (step 4) so that the subgraph isomorphism runs a specific procedure according to the result of the matching. If G2 is not large enough (step 4.1), the unrolling factor is increased and the algorithm goes to step 2. If spill code is necessary (step 4.2), new vertices representing the generated spill code are included in the DAG and the algorithm goes to step 3. If the matching procedure cannot perform a matching according to the time constraint (step 4.3), the algorithm splits up the DAG and goes back to step 3. Once the matching is found, LLVM performs a step for code emission with physical registers.

3 Preliminary Results

In this section we present some preliminary results of our integrated approach for scheduling and register allocation on the LLVM compiler. The results are presented in Table 1. The set of programs used for the experiment are part of Benchmark Simple of the Trimaran compiler [10].

Programs	Emitted Instructions	Allocated Registers
alloca_test	40	10
fact2	28	04
fib	45	09
fib_mem	37	13
ifthen	66	26
local_var_test	40	15
longlong	25	10
mm	110	36
struct_test	63	28
swtich_test	508	112

Table 1. Results of the Scheduling and Register allocation based on Subgraph Isomorphism.

Table 1 shows the number of emitted instructions and registers usage for each program. We have performed the same experiment (same programs and inputs) using the basic and greedy registers allocators and the fast scheduling algorithm available in the LLVM compiler (version 2.9 and 3.0). The combination fast+basic and fast+greedy algorithms provide the same number of instructions and registers of our technique. The fast+basic strategy has a performance better

than isomorphism strategy ranging from 10ms-25ms (average compilation runtime is 49ms for isomorphism and 37ms for fast+basic). No spill codes have been generated for all evaluated programs.

4 Final Remarks

An integrated approach for instruction scheduling and register allocation was presented in this paper. Our approach is based on matching input DAGs to the base graph of the architecture. The matching result is a subgraph of the base graph isomorphic to the input DAG that represents the allocated resources to run the DAG.

Despite not improving the performance compared to the list scheduling and basic register allocator algorithms, our approach seems promising for instruction scheduling and register allocation in the presence of heavy register and interconnection constrained architectures. For those cases, a technique that exposes all constraints and global resources seems to be useful to converge to better solutions faster than greedy algorithms.

Future work is focused on finishing up the implementation of the algorithm on the LLVM compiler to generate code for the ρ-vex processor [9] and evaluating it on well known benchmarks from compiler and computer architecture areas.

References

1. Chris Lattner, Vikram S. Adve: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. CGO 2004: 75-88 (2004)
2. Ricardo R. Santos, Rodolfo Azevedo, Guido Araujo: Instruction Scheduling Based on Subgraph Isomorphism for a High Performance Computer Processor. JUCS 14(21):34653480 (2009)
3. James R. Goodman, Wei-Chung Hsu: Code scheduling and register allocation in large basic blocks. ICS 1988: 442-452 (1988)
4. Shlomit S. Pinter: Register Allocation with Instruction Scheduling: A New Approach. PLDI 1993: 248-257 (1993)
5. Rajeev Motwani, Krishna V. Palem, Vivek Sarkar , Salem Reyen: Combining Register Allocation and Instruction Scheduling. CS-TN-95-22 (1995)
6. Cindy Norris, Lori L. Pollock: Experiences with Cooperating Register Allocation and Instruction Scheduling. Int. Journal of Parallel Programming 26(2): 241-283 (1998)
7. Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, Mario Vento: An Improved Algorithm for Matching Large Graphs. Proc. of the 3rd IAPR TC15 Workshop on Graph-Based Representations (2001)
8. David A. Patterson, John L. Hennesy: Computer Organization and Architecture: A Hardware/Software Interface. Addison Wesley. 2nd edition (2004)
9. Sthefan Wong, Thijs van As, Geoffrey Brown: -VEX: A Reconfigurable and Extensible VLIW Processor. ICFPT 2008 (2008)
10. Lakshmi N. Chakrapani, John C. Gyllenhaal, Wen-mei W. Hwu, Scott A. Mahlke, Krishna V. Palem, Rodric M. Rabbah: Trimaran: An Infrastructure for Research in Instruction-Level Parallelism. LCPC 2004: 32-41 (2004)