Instruction Assignment for Clustered Processors using Constraint Programming

Mirza Beg
mebg@cs.uwaterloo.ca
David R. Cheriton School of Computer Science
University of Waterloo

1 Problem and Motivation

A fundamental problem in compiler optimization, which has increased in importance due to the spread of multi-core architectures, is to find parallelism in sequential programs. Current processors can only be fully taken advantage of if workload is distributed over the available processors. In this work we look at distributing instructions in a block of code over multi-cluster processors, the *instruction assignment problem*.

We derive motivation to solve this problem from the proliferation of clustered processors in embedded systems. Since they are low cost and have low power consumption they are widely used in devices like cell phones, music players, cameras, printers and in the automotive marked for safety, engine control and navigation. Billions of these processors are sold each year and with the ever increasing demand and variety of consumer electronics the sales are likely to keep increasing.

Instruction scheduling on realistic processors is known to be a hard problem and compilers use heuristic approaches to schedule instructions. Instruction assignment can be simply stated as scheduling instructions on multiple processors. The idea would be to partition the DAG and schedule each partition on a processor. Figure 1 shows an example superblock and how it can be scheduled on a single cluster and a dual-cluster architecture.

The optimal assignment of instructions in blocks of code on multiple processors is known to be NP-complete. We present a constraint programming approach for scheduling instructions on multi-cluster processors that feature fast inter-cluster communication. We employ a problem decomposition technique to solve the problem in a hierarchical manner where an instance of the master problem solves multiple sub-problems to derive a solution. We found that our approach was able to achieve an improvement of 6%-20%, on average, over the state-of-the-art techniques on superblocks [8] from SPEC 2000 benchmarks.

2 Background and Related Work

The instruction assignment problem can be formally stated for code blocks as follows.

Definition 2.1 (Instruction Assignment Problem). Given the dependence graph $G = (V, E)$ for a basic block or a superblock and the number of available clusters k in a given architectural model, the *instruction assignment problem* is to find an assignment A and a schedule S such that $A(i) \in$

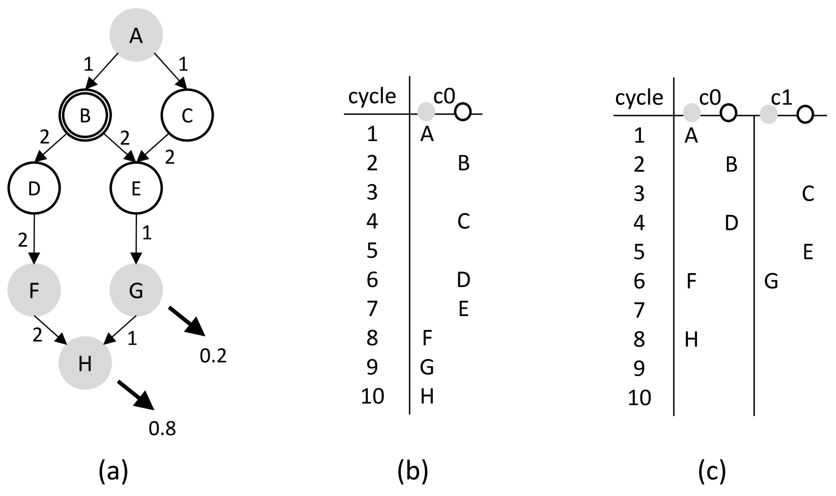

Figure 1: (a) Graph representation of a superblock: G and H are branch instructions with exit probabilities of 0.2 and 0.8 respectively. B is a serializing instruction and C is a non-piplined instruction. (b) A possible schedule for the superblock given in (a) for a single-cluster which is dual-issue and has two functional units. One functional unit can execute clear instructions and the other can execute shaded instructions. The weighted completion time for the schedule is $9\times0.2 + 10\times0.8 = 9.8$ cycles. (c) A possible schedule for the same superblock for a dual-cluster where the processors can communicate with unit cost and each processor is the same as the single-cluster in (b) The assignment of C, E and G to cluster $c1$ and the rest of the instructions to $c0$ results in a schedule with weighted cost of $6\times0.2 + 8\times0.8 = 7.6$ cycles.

$\{1, \ldots, k\}$ for each instruction i in the block and start time $S(i) \in \{1, \ldots, \infty\}$ for all the instructions $i \in V$ that minimizes the schedule length of the basic block or the weighted completion time in case of superblocks.

The most well known solutions to the assignment problem are greedy and hierarchical partitioning algorithms which assign the instructions before the scheduling phase in the compiler. The bottom-up greedy, or BUG algorithm [6] proceeds by recursing depth first along the data dependence graph, assigning the critical paths first. It assigns each instruction to a processor based on estimates of when the instruction and its predecessors can complete execution at the earliest. These values are computed using the resource requirement information for each instruction. The algorithm queries this information before and after the assignment to effectively assign instructions to the available processors.

Chu et al. [2] describe a region-based hierarchical operation partitioning algorithm (RHOP), which is a pre-scheduling method to partition operations on multiple processors. In order to produce a partition that can result in an efficient schedule, RHOP uses schedule estimates and a multilevel graph partitioner to generate cluster assignments. This approach partitions a data dependence graph based on weighted vertices and edges. The algorithm uses a heuristic to assign weights to the vertices to reflect their resource usage and to the edges to reflect the cost of inter-processor communication in case the two vertices connected by an edge are assigned to different processors.

In the partitioning phase, vertices are grouped together by two processes called *coarsening* and *refinement* [7, 9]. Coarsening uses edge weights to group together operations by iteratively pairing them into larger groups while targeting heavy edges first. The coarsening phase ends when the number of groups is equal to the number of desired processors for the machine. The refinement phase improves the partition produced by the coarsening phase by moving vertices from one partition to another. The goal of this phase is to improve the balance between partitions while minimizing the overall communication cost. The moves are considered feasible if there is an improvement in the gain from added parallelism minus the cost of additional inter-processor communications. The algorithm has been implemented in the Trimaran framework. Subsequent work using RHOP partitions data over multi-core architectures with a more complex memory hierarchy [4, 3].

3 Approach and Uniqueness

In this work, we present a constraint programming approach to instruction assignment on clustered processors that is robust and optimal. In a constraint programming approach, a problem is modeled by stating constraints on acceptable solutions, where a constraint defines a relation among variables, each taking a value in a given domain. The constraint model is usually solved using backtracking search. The novelty of our approach lies in the decomposition of the problem and our improvements to the constraint model, which reduces the effort needed in the search for the optimal solution. Our approach is applicable when larger compile times are acceptable. In contrast to previous work we assume a more realistic instruction set architecture containing non-fully pipelined and serializing instructions.

Each instruction is represented by a node in the basic block dependency graph. Each node i in the graph is represented by two variables in the model, x_i and y_i. The variable $x_i \in \{1, \ldots, \infty\}$ is the cycle in which the instruction is to be issued. The upper-bound to these variables can be calculated using a heuristic scheduling method. The variable $y_i \in \{1, \ldots, k\}$ identifies the cluster to which instruction i is assigned. The key is to scale up to large problem sizes. In developing an optimal solution to the assignment problem we have applied and adapted several techniques from the literature including symmetry breaking, branch and bound and structure based decomposition techniques.

The main idea is to solve the two stage problem of assignment and scheduling. The idea was inspired by integer programming, Benders [1] and Dantzig-Wolfe [5] decomposition techniques, to decompose the problem into master-slave. We model the assignment problem as master which models and solves multiple slave problems to schedule instructions for a given assignment at each stage. We extended an existing solver [11] to schedule the superblock for given assignments.

4 Results and Contributions

In our experiments we evaluated our approach on the SPEC 2000 integer and floating point benchmarks, using different architectural models. We compared our results against RHOP using the same communication cost as in [2]. We found that in our experiments we were able to improve on RHOP up to 79% depending on the architectural model. Also in our experiments we were able to solve a large percentage of blocks optimally with a ten minute time limit for each block. This represents a significant improvement over existing solutions.

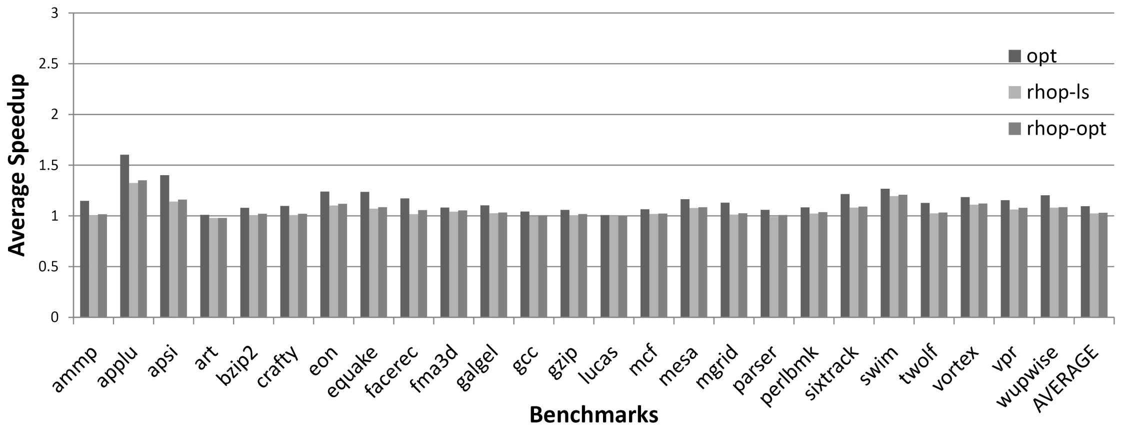

Figure 2: Average speedups of superblocks in SPEC 2000 for a 4-cluster 2-issue architecture.

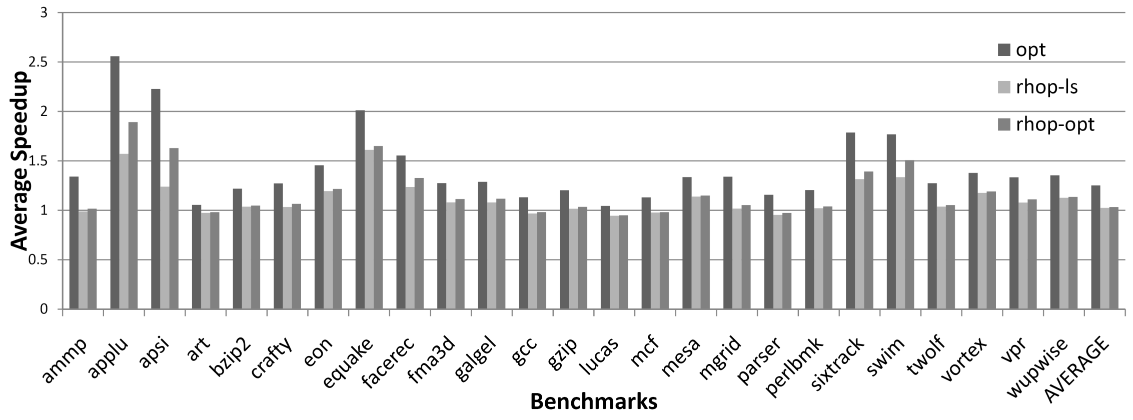

Figure 3: Average speedups of superblocks in SPEC 2000 for a 8-cluster 1-issue architecture.

We evaluated our instruction assignment algorithm with respect to how much it improves on previous approaches. The state-of-the-art instruction assignment algorithm for instruction assignment is a hierarchical graph partitioning technique known as RHOP [2] which uses coarsening and refinement to partition the dependency graph for multiple processors. Once the partitions are formed our simulation uses a list scheduler (using dependence height and speculative yield (DHASY) heuristic, which is also the default heuristic in the Trimaran compiler) to determine the schedule for the superblocks. To determine the effectiveness of our partitions the results are also reported for RHOP-OPT where the schedule for the given partitions is determined by an optimal scheduler [11]. The performance improvements are measured over the list scheduling algorithm for a single cluster architecture. For running our algorithm we used a timeout of ten minutes for each superblock. A ten minute timeout on each superblock allows an entire benchmark to be compiled in a reasonable amount of time.

Figures 2 and 3 compare the performance of a 4-cluster dual issue architecture using our algorithm(opt), RHOP (rhop-ls) and RHOP-OPT (rhop-opt) with the performance of 8-cluster single issue architecture. For the 4-cluster architecture our algorithm achieves improvements from 0.8% (lucas) upto 22.8% (apsi) over rhop-ls and from 0.8% upto 20.7% over rhop-opt. On average we improved over the previous techniques by 6%-7% for the 4-cluster configuration. In the case of the 8-cluster configuration our algorithm improves over the rhop-ls by 22% and over rhop-opt by 21% on average.

We have successfully applied constraint programming to the instruction assignment problem and made a strong case for automatic extraction of parallelism from programs in clustered processors. This will significantly improve the performance of embedded devices which are ubiquitous in today's consumer electronics.

References

[1] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik 4, 238-252, 1962.

[2] M. Chu, K. Fan and S. Mahlke. Region-based hierarchical operation partitioning for multicluster processors. PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pages 300-311, 2003.

[3] M. Chu and S. Mahlke. Compiler-directed data partitioning for multicluster processors. In CGO '06: Proceedings of the International Symposium on Code Generation and Optimization. pp 208–220, 2006.

[4] M. Chu, R. Ravindran and S. Mahlke. Data access partitioning for fine-grain parallelism on multicore architectures. In MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. pp 369–380, 2007.

[5] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. Operations Research 8, pages 101-111, 1960.

[6] J. R. Ellis. Bulldog: a compiler for VLSI architectures. MIT Press. Cambridge, MA, USA, 1986.

[7] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing. pp 28, 1995.

[8] W. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm and D. M. Lavery. The superblock: An effective technique for VLIW and superscalar compilation. The Journal of Supercomputing, 7(1), 229-248, 1993.

[9] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing. 201:359–392, 1998

[10] P. van Beek and K. Wilken. Fast optimal instruction scheduling for single-issue processors with arbitrary latencies. Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming, Paphos, Cyprus, 625-639, November, 2001.

[11] Tyrel Russell, Abid M. Malik, Michael Chase, and Peter van Beek. Learning heuristics for the superblock instruction scheduling problem. IEEE Transactions on Knowledge and Data Engineering, 21(10):1489-1502, 2009.