ABSTRACT
Instruction set architects and compiler writers often have conflicting requirements on the design of the instruction set. The architects are driven by potential micro-architecture complexity and possibly by code size considerations; while the compiler writers prefer clean orthogonal instruction sets amenable to regular compiler algorithms. This struggle has stabilized with a resulting uneasy boundary between the RISC and CISC architectural styles in general purpose computing. However, in the embedded computing world, with domain specific processors (e.g., Digital Signal Processors (DSPs)), this struggle continues. It has resulted in very irregular architectures that are very hard to compile for and that need specialized optimization techniques.

An increasing trend in embedded processing is to use VLIW (Very Long Instruction Word) processors, which need to be supported by efficient compilers that can extract the instruction level parallelism (ILP) at compile time. However, the ISAs (Instruction Set Architectures) for these machines are still driven by embedded processing constraints (small code size, simpler hardware), and consequently all available ILP may not always be cleanly reflected in the ISA. This presents a significant problem for the compiler, since tradition VLIW compilers work directly with physical resources to determine what can and cannot be scheduled in parallel. It is obviously desirable to leverage the large body of work (and software) in VLIW compilation to be directly used for these irregular processors.

The artificial resources are generated by solving a combinatorial graph labeling problem which is generated from the ISA specification. We show that this problem is equivalent to the problem of covering all the edges in a given graph using the minimum number of cliques (complete subgraphs). This is a known NP-complete problem - thus, we can leverage the heuristics already developed for this. In addition, there is a direct transformation between this problem and the well studied graph coloring problem. This transformation enables us to use the large body of work (and software!) in the graph coloring problem domain.

Finally, we demonstrate the application of these ideas in the development of compilers for two proprietary DSPs - the Elixir and the Hiperion DSPs from Fujitsu. In the past, compilers were developed for these processors using highly specialized optimization algorithms. This is now being replaced by using the proposed technique of generating artificial resources with an existing VLIW compiler (the IMPACT compiler).

We believe that the development of these ideas opens the door for using classic VLIW compilation methods for irregular embedded VLIW DSPs and is a significant step forward in the development of software tools for embedded processing.

1. INTRODUCTION
1.1 Motivation
Pressures to reduce the code size of VLIW (Very Long Instruction Word) machines force processor designers to design processors with irregular Instruction Set Architectures (ISAs). This is particularly true of Digital Signal Processors (DSPs) where this is driven by not just code size, but also by power considerations. This pressure forces designers to stray away from compiler friendly regular architectures to appli-
cation domain optimized and irregular micro-architectures. One of the key techniques used in DSPs (most of which have some VLIW capabilities) to reduce code size is to reduce the instruction width. This complicates the different phases of compilation and in particular the scheduler and the register allocator. Using a single instruction width to specify instructions with only one operation and those with multiple operations leads to “operation versioning” where the bit-crunch imposes different restrictions on register usage when the same operation is issued separately and when issued in parallel.

Another common feature in embedded processor technology to reduce code size is to prevent instructions from being as wide as the number of functional units would allow. In practice, all operations do not occur frequently. This makes it difficult to fill all operation slots every cycle, thus leading to an increased number of NOPs. At the same time, none of the functional units can be omitted from the design. A reduction in the number of ISA issue slots naturally leads to deciding which operations can be executed in parallel based on the set of applications for which the processor is targeted rather than on what the processor can offer in terms of resources. Again this results in significantly higher compiler complexity.

One possible optimizations to reduce area of a VLIW processor is to restrict the freedom of an instruction from being able to be issued in any ISA slot of the processor and instead to only certain fixed slots. Here all possible combinations of operations that can be executed in parallel must be specified explicitly keeping physical while resources in mind. This optimization reduces the decode logic required for each operation slot. Thus, in a significant number of the cases, the different constraints on combining operations that can be executed in parallel is restricted by more than just the ISA issue width or physical resources. In the worst case, compiler developers may not be aware of the actual micro-architectures that they are compiling for but know only the instruction set and the coding constraints on operation issue.

**Example**

This work was motivated by our development of a compiler for the Hiperion DSP using the IMPACT VLIW compiler [8]. We would like to highlight however, that the issues motivating this work are very general and are present in many ISAs, especially in the DSP domain.

The following is an example from the Hiperion processor. This processor allows the following combinations of operations: an ADD and SUB in parallel, an ADD and LOAD in parallel, a SUB and LOAD in parallel and a MUL and 1 or 2 LOADs in parallel. Clearly, from a functional unit point of view, it can execute an ADD, SUB, MUL and 2 LOADs in parallel, but this combination is not supported. Also mere physical resource specification would not do the job, as the issue width is variable, i.e. it is not enough to say that only 2 operations can be issued in parallel, since this is not always the case depending on the actual combination of operations. As mentioned before, the reduction of the instruction size in the Hiperion causes “operation versioning” of the LOAD, ADD and SUB. In particular LOADs use different addressing modes and destination registers when issued with an arithmetic instruction than when issued with another LOAD and arithmetic instruction.

If register allocation is performed after scheduling for such processors, splitting the operation into different versions lets each version be treated as a separate operation. While the register allocator does not have to change, the scheduler is faced with the additional burden of having to choose the best version from among different versions of the same operation. On the other hand, having a single operation for all versions relaxes the scheduler but increases the register allocator’s responsibility. The register allocator has to determine in which combination that operation has been scheduled. This means that this operation is no longer independent of the schedule and the register allocator requires additional instruction information to do its job.

### 1.2 Using VLIW Compilers

**Resource Based Schedulers**

Conventional resource based schedulers build a table of resources versus the cycle time and do a book-keeping based schedule once the data and control dependences are satisfied e.g. the scheduler described in reference [9]. For every operation issued, the scheduler updates the resource utilization table entries for the set of resources that the operation uses. So, two operations that can execute in parallel can be scheduled in the same cycle only if the resources are available. This property of the resource based schedulers prompted us to explore the possibility of converting irregular constraints into conventional resource constraints.

**Modeling Irregular Architectural Constraints with Regular Resource Constraints**

In this paper, we propose a method to convert irregular ILP constraints to regular resource based constraints and balance the register allocator and scheduler complexities for irregular architectures. We achieve this task by using the specifications of the irregular architecture to generate a set of regular resource constraints that can subsequently be used by conventional VLIW schedulers (e.g. IMPACT [8]). The resource constraints generated by our procedure may not correspond to any real physical constraints, and hence are artificial. However, they accurately capture the real ILP permitted by the ISA, and can be directly used in the VLIW scheduler.

This enables the large body of work (and software) used in conventional VLIW scheduling to be directly applied to irregular architectures. This avoids the development of specialized algorithms for supporting such architectures [10, 7, 1].

### 2. Problem Formulation and Solution

As mentioned in Section 1, we convert the Irregular ILP scheduling problem to a resource based regular VLIW scheduling problem by assigning artificial resources to each operation. We formulate the artificial resource assignment problem as the well known minimum clique edge covering problem. As input to the artificial resource assignment algorithm, we
take the set of all instructions supported by the ISA where each instruction is a set of operations that can be executed in parallel. We then construct a compatibility graph that captures the information about which pairs of operations can be executed in parallel. We show that a minimum clique edge cover of the complement of the compatibility graph produces a resource assignment to each operation such that the ILP constraints are satisfied and the total number of artificial resources used is minimum. To get the minimum clique edge cover, we convert the graph to its equivalent vertex clique partition problem and then use vertex coloring techniques to do vertex clique covering. In this section, we will describe the formulation and the algorithm in more detail.

2.1 Notation and Definitions

Here we introduce some notation and definitions. Some of the definitions and notation may seem superfluous for describing the problem statement, however, the additional material here will facilitate use of the same notation in Section 3 where we analyze shortcomings and assumptions in the algorithm and properties of reservation table based scheduling in general.

First we formalize the notion of operation types, operations, and resource usages.

**Definition 1 (Schedulable Operations).** We call the tuple \((Q, O, T)\) schedulable operations, where \(Q \subseteq I \times O\), \(O\) is a set of operation types, and \(T : Q \rightarrow O\) is a function where \(T((i, o)) = o\). When \(O\) and \(T\) are understood, we refer to \(Q\) as a set of operations. We may also use \(Q\) to refer to the entire tuple. For our purposes \(I\) can be any set.

**Definition 2 (Resource Usages).** We say \(A\) a defines resource usage alternatives for a set of operation types \(O\) if \(A : O \rightarrow 2^N\), \(R\) is some a set of resources and \(A = 2^{N \times R}\). We also call the members \(M(o)\) the resource usage alternatives for \(o\), and we call \(A\) the resource usage or resource alternative map. We also say that the elements \(A(o)\) provide resource assignments for \(o\). The elements of the resource usage alternatives are called resource usages.

Here we can consider the tuple \((n, r) \in N \times R\) to represent the need for resource \(r\), \(n\) cycles after the corresponding operation is issued. Each subset of \(N \times R\) \(A(o)\) is an alternative set of resources that can be used for operations of type \(o\).

When describing the graph optimization we will use the following terminology.

**Definition 3 (Minimum Clique Cover).** For a graph \(G\), the minimum set of cliques (complete sub-graphs) such that every edge of \(G\) is in at least one of these cliques. This problem is also called the Minimum Clique Edge Cover problem.

**Definition 4 (Minimum Clique Partition).** For a graph \(G\), the minimum set of cliques such that every vertex of \(G\) is in one and only one of these cliques. This problem is also known as the Minimum Clique Vertex Partition problem.

2.2 Graph formulation

As input we are given some ISA with the set of operation types \(O\) and a list of which operations may execute in parallel (exist in the same instruction). We build The compatibility graph \(G = (V, E)\), a simple undirected graph, constructed from the given input as follows: Each vertex \(v \in V\) in the graph corresponds to an operation type \(o \in O\). An edge \(e = (v_i, v_j) \in E\) exists between vertices \(v_i, v_j \in V, i \neq j\) if the corresponding operations \(o_i\) and \(o_j\) can be executed in parallel in at least one of the instructions in the ISA.

Figure 1 shows the compatibility graph for the example presented earlier.

![Figure 1: Compatibility graph for the Hiperion example.](image)

We want to assign resources to each operation type such that the resource usage map provides exact information to the scheduler as to what can and cannot be done in parallel. For this algorithm we assume that each operation has exactly one resource usage alternative. So, with the notation defined above, we wish to construct \(A\) which assigns resource usages to \(O\) such that for each \(o \in O\), \#\(A(o)\) = 1 and \(A\) satisfies the following constraints for any operation pair \(o_i, o_j \in O, o_i \neq o_j\):

**Constraint 1:** \(b(A(o_i)) \cap b(A(o_j)) = \emptyset\), if \(o_i\) and \(o_j\) can be executed in parallel and

**Constraint 2:** \(b(A(o_i)) \cap b(A(o_j)) \neq \emptyset\), if \(o_i\) and \(o_j\) cannot be executed in parallel.

where the function \(b\) simply extracts the only element in \(A(o)\) and maps it in the natural fashion to an element in \(2^{N \times R}\). These constraints simply state that if \(o_i\) and \(o_j\) can execute in parallel, they do not share any resources, and if they cannot execute in parallel, they must share a resource. Since we are only interested in simultaneous issue, all of our
resources can be assigned to cycle 0 of every operation type and thus the elements \( b(A(o)) \) always have the form \((0, r)\).

We represent the resource usages for each operation type as labels in the compatibility graph \( G \). Thus we need to solve the following labeling problem, subject to some constraints on the labels:

Assign to each vertex \( v_i \in V \), a non-empty set of labels \( L_i \) such that the cardinality of \( \bigcup_{v_i \in V} L_i \) is minimum. Each label thus created corresponds to an artificial resource and every set of labels \( L_i \) corresponds to a resource set \( R_i \) that is assigned with \( q_i \) in cycle 0, i.e. \( A(q_o) = \{(0, r) : r \in L_i\} \).

The motivation for minimizing the number of labels is that this directly minimizes the number of resources used in the VLIW scheduler. Fewer resources implies faster scheduling for many schedulers.

Once we have determined a set of artificial resources for an ISA, we build a machine description based on these resources and pass use it to compiler for our given ISA. For example, we could generate an HMDES[4] file based on the algorithm output and use it within the IMPACT scheduler[8]

3. ALGORITHM ANALYSIS

3.1 Correctness

Let \( G \) be defined as in Section 2.2. The following key result demonstrates the equivalence of our labeling problem to a well known combinatorial problem.

**Theorem 1.** Let \( C \) be a minimum clique edge cover of the edges of \( G \). For each clique, \( C_i \) in this cover in which a vertex \( v_i \) is contained, assign a label \( l_k \) to \( v_i \). The set of labels at \( v_i \) is \( L_i = \bigcup_{k : v_i \in C_k} l_k \). For each \( v_i \) such that no \((v_i, v_j) \in E_i \) we assign a unique label. This set of labels directly satisfy Constraints 1 and 2 and the number of labels is minimum.

**Proof:** An edge \( e = (v_i, v_j) \) in \( G = (V, E) \) represents the fact that operations \( O_i \) and \( O_j \) can be executed in parallel. In the complement graph, an edge between two vertices \( v_k, v_l \in V \) represents the fact that operations \( O_k \) and \( O_l \) cannot be executed in parallel and therefore must share a resource. A trivial solution that follows from this is to create a resource for each edge \( e' = (v_k, v_l) \). A clique \( C \) of \( G' \) represents the fact that no two operations represented by the vertices of \( C \) can be executed in parallel and consequently one resource shared by all operations corresponding to vertices of \( C \) is sufficient to prevent any subset of operations from executing in parallel. Hence a clique edge cover of all the edges in the complement graph satisfies Constraint 2 for every pair of distinct operation types \( o_i \) and \( o_j \). For operation types of the same type that have no incident edges on the corresponding vertex in \( G' \), we add a resource usage, and thus the operation type conflicts with itself and satisfies constraint 2. For operation types that have incident edges on their vertices in \( G' \), they clearly conflict with themselves. Since only distinct vertices connected by an edge in the complement graph share a resource by this method, two distinct vertices that are adjacent in \( G \) will not share a resource as there will not be any edge between them in the complement graph. Hence a clique edge cover of the complement graph also satisfies Constraint 1. We show in Section 3.2 that under the assumption that \( \#A(o) = 1 \), no operations of the same type may execute in parallel.

We now prove that a minimum clique edge cover produces a resource assignment using the minimum number of resources.

Let us assume that there exists another algorithm that produces \( L = \{l_1, l_2, ..., l_k\} \) as the set of labels from which each vertex \( v_i \) in \( G \) is assigned a set of labels \( L_i \subseteq L \) such that both constraints 1 and 2 are satisfied.

Consider the set of vertices \( V_i = \{v_i : l_i \in L_i\} \) Consider any two vertices \( v_p, v_q \in V_j \), \( l_j \in L_p \cap L_q \). Since this resource assignment satisfies Constraints 1 and 2, \( (v_p, v_q) \in E' \). Hence the set \( V_j \) forms a clique in \( G' \). Consider \( (v_p, v_q) \in E' \). \( L_p \cap L_q \neq \emptyset \). Hence there exists an \( l_j \in L_p \cap L_q \). Hence \( v_p, v_q \in V_j \). Therefore every edge in \( E' \) is included in some clique and \( V_1, V_2, ..., V_n \) form an clique edge cover of \( G' \).

Hence any valid label assignment of \( G \) will produce a clique edge cover of \( G' \). Therefore the minimum clique edge cover of \( G' \) will produce the label assignment with the minimum set of labels. Clearly, isolated vertices in \( G' \) can execute in parallel with any other operation type and thus cannot share resources with any other vertex and thus must be assigned their own resource. **QED**

The minimum clique edge cover problem is a known NP-complete problem (Problem GT17 in Garey and Johnson [3]). Thus, we can directly use known heuristics for that problem to solve our problem at hand. Specifically, Kou et al. [5] provide an heuristic algorithm for tackling this problem. In addition, in their proof of NP-completeness for this problem, they show how it can be converted to the classic graph vertex coloring problem. This enables us to directly use the large body of work developed for the graph coloring problem.

Figure 2 shows the complement compatibility graph for the Hiperion example and a resource assignment for the cliques. Figure 3 shows the labels assigned to vertices.

3.2 Applicability

The algorithm described above makes some implicit assumptions about the ISA. In this section we enumerate those assumptions and show how the relationships between them. The assumptions limit the applicability of the algorithm to only certain ISAs. We describe potential extensions to the algorithm in Section 5 to expand the applicability of our approach.

Let \( O \) be the the set of operation types in an ISA. Let \((Q, O, T)\) be schedulable operations. Let \( R \) be a set of resources potentially used by operations in the ISA. Let \( A \)
We will now show that assumption 1 and modeling operation define resource usages for the edges of the complement graph.

Figure 2: Complement compatibility graph for the Hiperion example with labeled cliques.

Figure 3: Compatibility graph for the Hiperion example with labeled vertices. The dashed edges show the edges of the complement graph.

define resource usages for $O$ such that $A(o) \neq \emptyset$. Let $A = 2^{N \times R}$.

The problem formulation makes the following assumptions on the ISA:

1. An operation has a single resource usage alternative, i.e. $\#A(o) = 1$, $\forall o \in O$.
2. An operation $(i, o)$, $o \in O$ cannot be issued in parallel with an operation of the same type.
3. If a set of operation types $P$ can be issued in parallel then all subsets of operation types in $P$ can be issued in parallel (and thus form a clique in $G$).
4. If sets of operation types $P$, which are subsets of operation types in $P$ can be issued in parallel, such that for any operation type pair $a, b \in P$, $(a, b) \in P$ for some $i$ then the operation types in any subset of $P$ can be issued in parallel, (and thus the set of vertices corresponding to the elements of $P$ form a clique in $G$).

We will now show that assumption 1 and modeling operation conflict with resources implies assumptions 2, 3, and 4. In order to show this, we will need a few more definitions. We discuss relaxation of some of the assumptions in Section 5.

**Definition 5 (Schedule).** We call $S$ a schedule if $S = (f, (Q, O, T), A)$ where $f : Q \rightarrow N$ is a function, $(Q, O, T)$ are schedulable operations, and $A$ defines resource usages for $O$.

Here $f$ simply defines the cycle in which an operation in $Q$ is issued, and $A$ defines the set of possible resource usages for each operation type as we saw before.

**Definition 6 (Resource Alternatives).** Given a schedule $S = (f, (Q, O, T), A)$, let $\text{shift}_n$ be an operator that, given $F \subseteq A$, for each element $a \in A(q)$ $\text{shift}_n F$ has an element $b$ such that $b = \{(j, r) \in N \times R : ((j - n), r)\}$.

We call $A_S$ the resource alternative map for $(Q, O, T)$ (or simply $Q$ if $(Q, O, T)$ is understood) in $S$ if $A_S : Q \rightarrow 2^A$ is a function such that $A_S(q) = \text{shift}_{T(q)} A(T(q))$. We call the members of $A_S$ the resource alternatives for $Q$ in $S$.

The function $A_S$ simply shifts all the resource usages in the members of $A(T(q))$ based on which cycle $q$ is issued.

**Definition 7 (Bound Schedule).** $S_b = (S, b)$ is called a bound schedule for $S = (f, Q, O, T, A)$ if $f : Q \rightarrow A$ is a function such that $b(q) \in A_S(q)$ where $A_S(q)$ is the resource alternative map for $Q$ in $S$.

A bound schedule is simply a schedule where we have selected the particular resource usage alternative for each operation that is in the schedule.

**Definition 8 (Valid Schedule).** A schedule $S = (f, (Q, O, T), A)$ is called a valid schedule if and only if there exists some $b$ such that $S_b = (S, b)$ is a bound schedule and $b(q) \bigcap \bigcup_{z \in Q, z \neq q} b(z) = \emptyset$.

A valid schedule is simply a schedule for which there is some binding of resource alternatives such that there are no resource conflicts.

Now that we have defined these few new terms, we are ready to show that assumption 1 implies all the other algorithm assumptions. Assumption 2 easily follows from assumption 1 provided that the operation type occupies some resources. Clearly, operations must occupy some resources, else the ISA would support execution of infinitely many of them. The assumptions are easy to verify given the following two theorems. The theorems themselves are fairly easy to prove with the defined notation.

**Theorem 2 (Subset Theorem).** Given a valid schedule $S = (f, (Q, O, T), A)$, and $q \subseteq Q$, there exists a valid schedule $S' = (f', Q', O, T, A)$ where $Q' = Q - q$, and $f' : Q' \rightarrow N$ is a function such that $f'(z) = f(z), \forall z \in Q'$.
Proof: Since $S$ is valid there exists a bound schedule $S_b = (S, b)$ such that $b(n) \bigcap \bigcup_{z \in Q, z \neq n} b(z) = \emptyset, \forall n \in q$. But $\bigcup_{z \in Q', z \neq n} b(z)$ so $S_b = (S', b')$ where $b' : Q' \mapsto A$ such that $b'(z) = b(z), \forall z \in Q'$, is a bound schedule where $\forall n \in Q', (b(n) \bigcap \bigcup_{z \in Q', z \neq n} b(z) = \emptyset$. QED.

We see that Theorem 2 easily translates to assumption 3. Notice however, that Theorem 2 does not depend on assumption 1. This means that any ISA that violates assumption 3 cannot be scheduled using the traditional resource table based scheduling techniques. Furthermore, notice that when we apply Theorem 2 to the operations given in Theorem 2, we see that every ISA, having one resource usage alternative, yields a bound schedule and each schedule $S$ maps every $f \in A$ to graph coloring. We used a public domain utility developed for graph coloring [6] to perform the actual graph coloring.

The HIPERION DSP and ELIXIR DSP Instruction Set Architectures are used to test our algorithms. The results are summarized in Table 1. The column labeled Stat represents the statistics that we are interested in evaluating. Resources represents the number of artificial resources used to represent the constraints. $G = (V, E)$, $G' = (V, E')$ and $G_1 = (V, E)$ represent the Compatibility graph, Complement of the compatibility graph and the final graph used in the coloring algorithm, respectively. The major columns HIPERION and ELIXIR are the results of running the algorithm for the respective processors. Minor columns w ver. heur., w/o. ver. heur., w ver. exact and w/o ver. exact represent the statistics for the heuristic algorithm with and without considering different versions of instructions based on register allocation, and the exact coloring algorithm with and without considering register versioning respectively. The complexity of the labeling algorithm and thus our algorithm depends on the number of vertices and edges in the original graphs[5].

Since machine descriptions are usually generated infrequently the CPU run times don't matter too much as long as they are within reason. The results should be compared based on the total number of resources generated and the size of the graphs they generate. If the graphs become too large and exceed available memory, the exact algorithm becomes unusable, and the heuristic technique becomes more attractive. On the other hand, as long as run times for the exact method are within acceptable limits, it is the preferred method for generating the minimal resource set.

5. ALGORITHM EXTENSIONS

As we have seen from Section 3, in order to extend the results of our algorithm to a more general class of ISAs we need to relax assumption 1, and generate scheduling alternatives for each operation type, instead of a fixed set of operations. Furthermore, a particular implementation of an ISA may be partially described by explicit constraints on parallel execution and implicit constraints based on resource usage. In this section we briefly explore how to handle each of these situations.

First, we address the issue of the ISA described with resource constraints and explicit parallel execution constraints. We can address this problem simply by merging the results of our algorithm with a more traditional machine description.

First, the designer inputs a machine description based on the resource constraints posed by the implementation. Next, the designer inputs the issue constraints explicitly specified by the ISA manual. Once this is done, the designer can run our algorithm to generate a set of artificial resources. These artificial resource usages can then be added to the traditional machine description in every resource usage alternative for the appropriate instructions. Of course, this approach does not guarantee a minimal resource allocation for the machine, but the results can be run through a machine description optimizer such as the one described in reference[2].

Next, we deal with relaxing assumption 2. If the ISA supports the simultaneous issue of $n$ operations of operation
We believe that the development of these ideas opens the door for using classic VLIW compilation methods for irregular embedded VLIW DSPs and is a significant step forward in the development of software tools for embedded processing.

7. REFERENCES


Finally, we discuss some limitations of resource based scheduling, extensions to our algorithm and future research directions to cover all irregular ISAs supportable by resource driven schedulers.