An Optimistic and Conservative Register Assignment Heuristic for Chordal Graphs

Philip Brisk Ajay K. Verma Paolo Ienne
Processor Architecture Laboratory
Ecole Polytechnique Federale de Lausanne (EPFL)
Lausanne, Switzerland, CH-1015
{philip.brisk, ajaykumar.verma, paolo.ienne}@epfl.ch

ABSTRACT

This paper presents a new register assignment heuristic for procedures in SSA Form, whose interference graphs are chordal; the heuristic is called optimistic chordal coloring (OCC). Previous register assignment heuristics eliminate copy instructions via coalescing, in other words, merging nodes in the interference graph. Node merging, however, can not preserve the chordal graph property, making it unappealing for SSA-based register allocation. OCC is based on graph coloring, but does not employ coalescing, and, consequently, preserves graph chordality, and does not increase its chromatic number; in this sense, OCC is conservative as well as optimistic. OCC is observed to eliminate at least as many dynamically executed copy instructions as iterated register coalescing (IRC) for a set of chordal interference graphs generated from several Mediabench and MiBench applications. In many cases, OCC and IRC were able to find optimal or near-optimal solutions for these graphs. OCC ran 1.89x faster than IRC, on average.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors – *compilers*.

General Terms

Algorithms, Performance.

Keywords

Register Assignment, Static Single Assignment (SSA) Form, Chordal Graph.

1. INTRODUCTION

Register allocation is one of the most challenging compiler optimizations from both a theoretical and practical standpoint. Recent work in register allocation has focused on using the *Static Single Assignment (SSA)* form as an intermediate representation. The primary motivation for this is a recent proof that the interference graph for an SSA Form is a chordal graph [1, 6, 15]; although chordal graphs can be colored optimally and efficiently in polynomial time [11], a coloring solution alone does not yield an effective solution for register assignment.

Register allocation in compilers consists of two separate sub-problems: *spilling* and *register assignment*; this paper focuses on the latter. Spilling partitions all of the scalar variables that are live at each point in the program between the k registers available in the target architecture and memory, which is much costlier to access. Register assignment must map each of the non-spilled variables to the k target registers, with the goal of eliminating as many dynamically executed copy instructions as possible. A copy instruction between two variables can be eliminated by assigning both of the variables to the same register; the resulting instruction copies the content of the register to itself, and thus does not change the state of the processor, and can therefore be eliminated.

One important advantage of using SSA Form is that all of the copy instructions in an application can be eliminated while SSA Form is constructed [4]. After translating out of SSA Form, the only remaining copies are those that are inserted to replace φ-functions, an integral part of the SSA representation. Although it has never been formally proven or empirically observed that SSA-based register allocation is superior to traditional methods, the ability to eliminate the vast majority of copies up front appears to be a conceptual advantage at the very least.

To date, one SSA-based register allocation algorithm has been published by Hack et al. [16]. This allocator performs spilling and coalescing in two separate steps, unlike prior non-SSA-based graph coloring allocators (e.g. [5, 8, 12, 22, 28]). The spilling phase is similar in principle to linear scan register allocation [23], but traverses the program's dominator tree instead; register assignment, in contrast, is based on graph coloring. It is our belief that future SSA-based register allocators will also separate spilling and coalescing into distinct phases, unless just-in-time (JIT) constraints dictate that both be solved in conjunction.

One of the key requirements of register assignment is that no more than k colors can be used to color the interference graph. In order to minimize copies, the vast majority of prior register assignment heuristics employ a technique called coalescing. To eliminate a copy $y \leftarrow x$, the nodes in the interference graph corresponding to x and y are merged. Node merging, however, can increase the chromatic number of the interference graph. Conservative coalescing techniques [5, 12, 17-18] prevent coalescing unless it can be proven that doing so will not increase the chromatic number beyond k. Although the spilling phase ensures that the resulting interference graph is k-colorable using a simple greedy algorithm, it is generally NP-Complete to determine whether the graph remains k-colorable after coalescing even a single copy [3].

In SSA Form, however, the interference graph is chordal. The chromatic number of a chordal graph $G = (V, E)$ can be computed in $O(|V| + |E|)$ time using a simple greedy algorithm [11]; however, coalescing cannot preserve the chordal graph property unless additional interference edges are added. To address this need, this paper introduces a new coalescing heuristic, *Optimistic Chordal Coloring (OCC)* for chordal graphs that does not employ coalescing and ensures that a legal k-coloring is found.

We have tested the OCC heuristic on a set of interference graphs generated from embedded applications taken from the Mediabench [20] and MiBench [14] benchmark suites. We compared OCC with *Iterated Register Coalescing (IRC)* [12], an established technique for register assignment for interference graphs. We also compare the results with an optimal algorithm based on *Integer Linear Programming (ILP)* [13]. We report the following results:

- For all but 3 applications, both OCC and IRC found solutions of optimal cost. OCC found a better result that IRC in only one case. We caution that this result is limited to a few embedded applications and is not necessarily representative of other classes of applications (e.g. scientific, general-purpose, etc.).
- For all but 3 applications, OCC ran faster than IRC; on average, OCC ran 1.89x faster than IRC. The 3 applications where IRC was faster generally had small interference graphs with relatively few φ-functions. Two of these three applications featured optimal solutions with 0 copies, indicating that conservative coalescing eliminated every copy. The more complicated techniques employed by OCC worked against its runtime in these cases.

The paper is organized as follows. Section 2 introduces SSA Form, chordal graphs, and defines the problem of register assignment. Section 3 summarizes related work in the area of register allocation, focusing specifically on coalescing techniques. Section 4 describes the OCC heuristic in detail, and compares its execution to prior heuristics based on coalescing. Section 5 presents our experimental evaluation of the OCC heuristic, which is compared to both the IRC heuristic and the optimal ILP solution for register assignment. Section 6 concludes the paper.

2. PRELIMINARIES

2.1 SSA Form

Any instruction of the form $x \leftarrow \ldots$ is a *definition* of variable x; any instruction of the form $\ldots \leftarrow x$ is a *use* of x. A procedure is defined to be in *Static Single Assignment (SSA)* Form [4, 9-10] if every variable is defined once and each use of a variable corresponds precisely to one definition. Fig. 1 shows a CFG fragment (a) converted to SSA Form (b). Variable x is defined on both sides of a condition and used at a block following the condition. The definitions of x are renamed to x_1 and x_2, in Fig. 1 (b); a φ-function is introduced to merge x_1 and x_2 into x_3, and the use of x is replaced with a use of x_3.

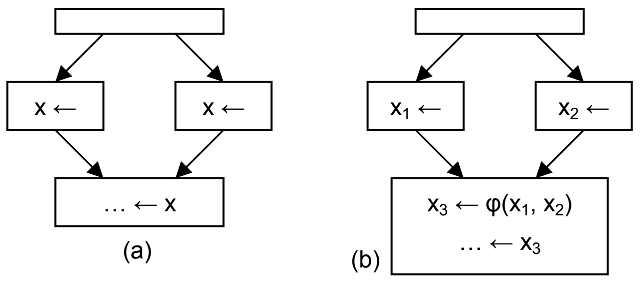


Figure 1. A CFG fragment (a) in SSA Form (b)

In Fig. 1 (b), if the left path is taken, then x_3 receives the value of x_1; if the right path is taken, then x_3 receives the value of x_2. φ-functions are always placed at the beginning of a basic block and execute in parallel.

Following register assignment, we translate out of SSA Form using *swap* and *parallel copy* instructions, as described by Hack et al. [15]. If swaps and parallel copies are not available in the target architecture, then the *Swap Problem* [4] may occur. Hack et al. suggested that 3 *xor* operations can be used to implement a swap. This approach generally works, but at a higher cost (i.e. 3 single-cycle instructions per swap compared to 1).

Unfortunately, some processors have dedicated registers for floating point values. The *xor* instruction may not be able to access these registers. In this case, 3 copy instructions are required; however, if no register is free, then some variable must be spilled in order to free up a register. The choice of which variable to spill in such a circumstance is beyond the scope of this paper; we simply assume that swap instructions are available.

The *Lost Copy Problem*, which is also beyond the scope of this paper, can be rectified by critical edge splitting [2, 4].

2.2 Chordal Graphs

Let $G = (V, E)$ be an undirected graph. For $v \in V$, $N(v)$ is the set of vertices adjacent to v. An *Elimination Order (EO)* is a function $\sigma : V \rightarrow \{1, 2, \ldots, |V|\}$ that assigns a unique value to each vertex. The vertices are named such that for vertex $v_i \in V$, $\sigma(v_i) = i$. Let $V_i = \{v_j \in V | j \leq i\}$, and $G_i = (V_i, E_i)$ be the subgraph of G induced by V_i. G_0 is defined to be the empty graph and $N_i(v) = N(v) \cap V_i$.

The basic idea of an elimination order is that graph G_{i+1} can be constructed from G_i by adding v_{i+1} to V_i and adding all edges belonging to $N_i(v_{i+1})$ to E_i. A *Perfect Elimination Order (PEO)* is an elimination order that sastisfies the property that $N_i(v_i)$ is a clique in G_i. A *Chordal Graph* is any graph that has a PEO. An interference graph for an SSA Form program is provably chordal [1, 6, 15].

A PEO for a chordal graph can be computed in $O(|V| + |E|)$ time using *Maximum Cardinality Search (MCS)* [27], whose pseudocode is shown in Fig. 2(a).

Given a PEO, an optimal color assignment can be computed for a chordal graph in $O(|V| + |E|)$ time using Gavril's algorithm shown in Fig. 2(b) [11]. Inductively, assume that a legal coloring has been found for vertices $v_1, \ldots, v_{i-1}$, now consider v_i. The only vertices that constrain the color assigned to v_i are those in $N_i(v_i)$, which is a clique. Therefore, it suffices to assign the smallest color to v_i that is not assigned to a vertex in $N_i(v_i)$.

```
Algorithm: Maximum Cardinality Search
Input: Chordal Graph G = (V, E)
Output: PEO σ : V → {1… |V|}
1 ∀ v∈V:   Let T(v) ← 0; σ(v) ← ϕ
2. For Integer : k ← 1 to |V|
3.     Select v∈V ∋ T(v) is maximum
4.     σ(v) ← k
5.     ∀ u∈N(v) ∋ σ(v) = ϕ :   T(u) ← T(u) + 1
6. EndFor
```

(a)

```
Algorithm: Chordal Color Assignment
Input: Chordal Graph G = (V, E), PEO σ
Output: Color Assignment f : V → {1… χ_G}
1 For Integer : i ← 1 to |V| in PEO order
2.     Let c be the smallest color not assigned
           to a vertex in N_i(v_i)
3.     f(v_i) ← c
4. EndFor
```

(b)

Figure 2 Pseudocode for Maximum Cardinality Search [27] (a) and Chordal Color Assignment [11] (b)

Gavril's algorithm is not the only algorithm that can optimally color a chordal graph. A heuristic called *Simplification* [19, 21], a variant of which has been used in the past for graph coloring register allocation [5, 8, 12, 22, 28], can also produce optimal colorings for chordal graphs [2]. Simplification repeatedly removes the vertex of minimal degree from the graph and pushes it onto a stack. The top-to-bottom order of vertices in the stack is then treated as an EO. Color assignment then proceeds using the same basic greedy approach as Gavril—assign each vertex that smallest color not assigned to one of its neighbors.

2.3 Problem Statement

Let $G = (V, E)$ be a k-colorable interference graph for a procedure in SSA Form, where k is the number of registers in the target architecture; since spilling has occurred already, the assumption that G is k-colorable is reasonable, and since the procedure is in SSA Form, G is chordal. Consequently, $\chi(G)$, the chromatic number of G, can be computed in $O(|V| + |E|)$ time, and $\chi(G) \le k$.

Now, consider a generic φ-function, $y \leftarrow \varphi(..., x, ...)$. Let r_y and r_x be the registers assigned to y and x respectively. If $r_y \ne r_x$, then a copy $r_y \leftarrow r_x$ must be inserted on the CFG edge leading into the block containing the φ-function. On the other hand, if $r_y = r_x$, then the copy instruction is an identity operation that does not change the state of the machine; it can therefore be eliminated.

Runtime profiling is used to determine the execution frequency of basic blocks, and also the number of times each CFG edge is taken. If runtime profiling information is not available, a static profile can be generated based on loop nesting depth. If the CFG edge that supplies x as a parameter to the φ-function is taken t times, then eliminating the copy shaves t processor cycles off the total execution time of the application (assuming a 1-cycle copy instruction and a single-issue processor).

The goal of the register assignment problem is to eliminate as many dynamically executed copies as possible while still ensuring that the assignment of variables to registers is legal. For our purposes, we introduce an *extended interference graph* $G = (V, E, E_\varphi, w)$, where w is a weight function that assigns integer weights to the *φ-edges* in E_φ based on profiling information; E_φ is defined as follows:

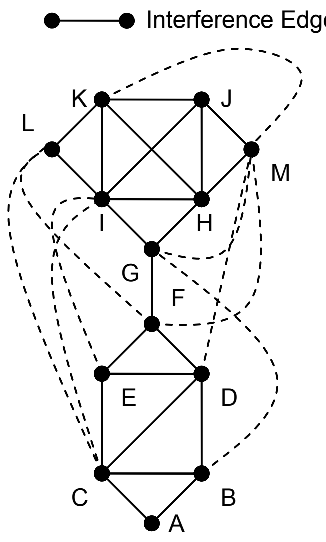
$$E_\varphi = \{(x, y) | x \ne y \wedge x, y \notin E \wedge \exists y \leftarrow \varphi(..., x, ...)\} \quad (1)$$

For φ-edge $e = (x, y)$, $w(e)$ is the dynamic execution count of the CFG edges leading into the basic block containing the φ-function defining y, where x occurs in the parameter slot.

Fig. 3 shows a sample problem instance: an interference graph, including the weighted φ-edges. The weights correspond to the number of times a copy instruction would execute if two vertices incident on the same φ-edge are assigned different colors. Any legal coloring of G that uses at most k colors is a solution to the problem of register assignment for a procedure. The problem can be solved individually for each procedure.

The efficacy of a coloring solution is the number of dynamic copies removed based on the estimated profile, w, which is derived from the color assignment. If f is a legal color assignment of G, we define an objective function, F, which computes the number of dynamically executing copies that will be eliminated:

$$F(G, f) = \sum_{\substack{e=(x,y)\in E_\varphi \\ f(x)=f(y)}} w(e) \quad (2)$$



φ-Edge	Weight
(B, G)	3
(C, I)	2
(C, L)	5
(D, M)	7
(E, I)	3
(F, L)	4
(F, M)	5
(G, M)	3
(K, M)	2

Figure 3. A sample problem instance: an interference graph with weighted φ-edges

Table 1.
Two color assignments for the graph in Figure 1.

Node	Color (1)	Color (2)
A	1	1
B	2	3
C	3	2
D	1	4
E	2	1
F	3	3
G	1	4
H	2	2
I	3	1
J	4	3
K	1	4
L	2	2
M	1	4
	20 copies	14 copies

The optimal color assignment, *f**, is the legal color assignment that uses at most *k* colors such that $F(G, f^*) \geq F(G, f)$ for every legal color assignment *f*. The decision analogue of this problem is NP-Complete, even for chordal interference graphs [3, 24].

This interference graph is chordal, and its chromatic number is 4. If application consists of one procedure, then 4 registers are allocated. Any legal 4-coloring of the interference graph suffices; however, there are many 4-colorings, each of which will cause a different number of copies to execute dynamically. Table 1 shows two legal 4-colorings. The first executes 20 copies dynamically; the second executes 14, and is a better solution.

3. RELATED WORK

This paper was motivated, in part, by a recent theoretical study of the complexity of register assignment via coalescing due to Bouchez et al. [3]. In particular, Bouchez et al. identified 4 categories of coalescing, as discussed in subsections 3.1-3.4.

It is important to note that to coalesce a copy $y \leftarrow x$, the nodes in the interference graph corresponding to *x* and *y* are merged into a new node, *xy*. The OCC heuristic, contrast, does not employ node merging, but retains the benefits of coalescing.

3.1 Aggressive Coalescing

Aggressive coalescing recklessly eliminates copy instructions without regard for the effect on the chromatic number of the interference graph. The first graph coloring register allocator by Chaitin [8] employed aggressive coalescing; Briggs et al. [5] later showed that this approach can cause additional spilling, and introduced conservative coalescing, which is discussed in subsection 3.2. Today, aggressive coalescing is most often studied under the context of translating out of SSA Form while minimizing the number of copies inserted while doing so [7, 24, 26]; it is also used as a phase of optimistic coalescing, which is discussed in Section 3.3. Bouchez et al. [3] showed that aggressive coalescing is NP-Complete, even for graphs containing at most 3 interference edges.

3.2 Conservative Coalescing

Conservative coalescing recognizes that load and store instructions, which are required for spilling, are generally more costly than copy instructions. Eliminating a copy is generally a good idea, but the effectiveness of doing so will cause further spills. Conservative coalescing criteria only allow copies to be coalesced if it can be proven that doing so will not cause additional variables to be spilled. Conservative coalescing criteria have been proposed by Briggs et al. [5], George and Appel [12], and Kaluskar [18]. The theoretical power of different versions of the Briggs et al. and George and Appel criteria has been studied in greater detail by Hailperin [17]. Bouchez et al. [3] have shown that conservative coalescing is NP-Complete for chordal graphs.

The limitations of conservative coalescing were well-known to Briggs et al. [5], who introduced the concept. In many cases, a copy $y \leftarrow x$ could not be coalesced due to a large number of neighbors of high degree; however, depending on the assignment of colors to their neighbors, it still may be possible to assign the same color to *x* and *y* during simplification, without node merging. This approach is called *biased coloring*.

3.3 Optimistic Coalescing

Park and Moon [22] and Vegdahl [28] observed that coalescing can have positive effects at well. Park and Moon introduced *optimistic coalescing*, which performs aggressive coalescing until no further coalescing is possible due to interferences. Some of the copies are then *decoalesced* when doing so can reduce spilling.

3.4 Incremental Coalescing

Both the aggressive and conservative approaches consider one copy instruction at a time for coalescing; this is called *incremental coalescing*. Bouchez et al. [3] identified situations where incremental coalescing yields sub-optimal results. Using conservative coalescing as an example, Bouchez et al. found an interference graph, where there are 2 copies, neither of which satisfies the criteria for conservative coalescing; however, if both copies are coalesced together, the criteria would be satisfied. Coalescing one copy at a time, unfortunately, can not achieve the best result. Unfortunately, if an interference graph contains *k* copies, then there are 2^k different subsets of copies that can coalesced in conjunction. Unless it is proven that $P = NP$, the optimal solution can only be found by coalescing every possible combination of copies and testing whether or not the resulting interference graph is *k*-colorable. Clearly, doing so would be prohibitively expensive. An analogous situation occurs in the context of decoalescing in the optimistic strategy.

4. OPTIMISTIC CHORDAL COLORING

The *Optimistic Chordal Coloring (OCC)* heuristic is influenced by previous coalescing techniques, but does not explicitly merge nodes. This ensures that the interference graph remains chordal and *k*-colorable, which satisfies the criteria for conservative coalescing. At the same time, OCC attempts to mimic the use of aggressive coalescing in the optimistic strategy. OCC identifies independent sets of nodes to which it would like to assign the same color; however, OCC does *not* coalesce these nodes. During color assignment, which processes nodes in PEO order, OCC makes every effort to ensure that all nodes in the same independent set receive the same color. It is not always possible to do so, however; when this occurs, the situation is analogous to decoalescing, as discussed in Section 3.3.

The identification of independent sets of nodes that would be coalesced in the context of aggressive coalescing, but not coalesced by OCC, is called *pseudo-coalescing*. Pseudo-coalescing is the key feature that sets the OCC heuristic apart from other heuristics based on coalescing.

Subsections 4.1-4.5 describe the different phases of our implementation of the OCC heuristic in detail.

4.1 Simplification

Let $G = (V, E)$ be a K-colorable graph and $v \in V$. If $|N(v)| < K$, then $G' = (V', E')$ is K-colorable, where $V' = V - \{x\}$ and E' is the subgraph of G induced by V'. In short, even if every neighbor of v is assigned a unique color, there is still at least one free color available for v [19]. Consequently, v can be removed from G, the resulting graph G' can then be colored, and a color can then be assigned to G after the fact. Many graph coloring register allocators [5, 8, 12, 22, 28] employ some type of simplification,

This is the use of simplification in the context of register allocation that was alluded to at the end of Section 2.2. The difference is that in Section 2.2, simplification was used as a heuristic to find the optimal coloring of a chordal graph (it can also act as a heuristic to find colorings of general graphs, with no guarantees of optimality).

Simplification is not necessary to compute a minimum coloring of a chordal graph; however, simplification is still useful for register assignment on chordal graphs as long as it is restricted to vertices having no incident φ-edges. Suppose that vertex v is incident on several φ-edges. If vertices that interfere with v occur prior to v in the PEO, then the colors assigned to these vertices constrain the color that can be assigned to v. By removing these vertices by simplification, fewer constraints are placed upon v at the time its color is assigned, and it is therefore more likely that color assignment can eliminate more copies involving v.

In the context of coloring chordal graphs for register assignment, Grund and Hack [13] used simplification to reduce the number of vertices that must be colored using their ILP. After coloring the remaining subgraph, colors are assigned to all vertices that were removed earlier by simplification. We essentially do the same thing here, but employ the OCC heuristic for coloring rather than the optimal ILP.

4.2 Biased MCS

In general, any technique that assigns colors to vertices incident on *φ*-edges earlier rather than later during the coloring process will lead to fewer dynamic copies than an alternative that assigns colors to them later. Although simplification is effective, it is not the only method that can achieve this. A chordal graph may have many different PEOs, an the PEO produced by MCS [27] in Fig. 2(a) is not particularly favorable in this respect.

We make the following modification to the MCS algorithm. During each iteration, let S be the set of vertices v such that $T(v)$ is maximum (line 3 of Fig. 2(a)). For vertex $s \in S$, let $W^*(s)$ be the sum of the weights of the φ-edges incident on s. When there is a choice between multiple vertices with maximum T-values, W^* is used as a tiebreaker, and the vertex with the maximal W^* value is chosen. This approach generally places vertices with high W^* values as early as possible in the PEO. This allows these vertices to be assigned colors before many of their neighbors, which means fewer constraints on the number of available colors. This approach is called *Biased Maximum Cardinality Search (BMCS)*.

4.3 Aggressive Pseudo-Coalescing

Pseudo-coalescing is similar in principle to coalescing: both identify an independent set of nodes in the interference graph that "should" be assigned the same color. Coalescing merges these nodes into a single node, therefore enforcing the decision regarding coloring; pseudo-coalescing, in contrast, simply retains the independent sets. The color assignment heuristic, discussed in subsection 4.4, then attempts to assign each node in the same independent set the same color. This approach is conservative for chordal graphs, because the decision *not* to merge nodes ensures that the chromatic number of the interference graph does not exceed k.

Let $S(v)$ be the independent set to which node v belongs. Initially, $S(v) = \{v\}$ for every vertex. Let $N(S(v)) = \{N(x)|x \in S(v)\}$. In other words, $N(S(v))$ is the set of nodes that are adjacent to at least one vertex in $S(v)$. Two distinct vertices u and v can be pseudo-coalesced if $N(S(u)) \cap S(v)$ and $S(u) \cap N(S(v))$ are both empty; if so, then the two sets are merged.

Pseudo-coalescing is effectively the same as coalescing, but without the node merging. If u is coalesced into some node c_1 and v is coalesced into c_2, then $N(S(u)) \cap S(v)$ and $S(u) \cap N(S(v))$ are both empty in the context of pseudo-coalescing if and only if c_1 and c_2 are not adjacent in the interference graph under coalescing.

The aggressive pseudo-coalescing phase in OCC is incremental [3], meaning that one φ-edge is pseudo-coalesced at a time. We sort the φ-edges by weight, and consider them for pseudo-coalescing in *Decreasing Sorted Order (DSO)*. Except for node merging, aggressive pseudo-coalescing is exactly the same problem as aggressive coalescing: the goal is to maximize the sum of the weights of the φ-edges that have been (pseudo-)coalesced. Since this problem is NP-Complete [3], considering φ-edges in DSO order is a heuristic, and does not guarantee optimality.

4.3.1 Paths and Clique Rays

Grund and Hack [13] identified two commonly occurring subgraphs in register assignment problems, shown in Fig. 4 (a) and (b),, and tailored their ILP formulation to exploit these types of subgraphs, when found. Specifically, they used the existence of such subgraphs to impose extra constraints (*cuts)* on the ILP which increase its rate of convergence. Here, we show that the DSO order can solve the aggressive coalescing problem optimally for these specific subgraphs.

Fig. 4(a) shows a *clique ray*: a subgraph that consists of a clique $C = \{v_1, \ldots, v_n\}$ in the interference graph, and a node $a \notin C$, such that there is a φ-edge (a, v_i) for $1 \leq i \leq n$. Clearly, $S(a)$ can only contain one node in C. If there are no other φ-edges incident on any vertices in a clique ray, then DSO will process the maximum weighted φ-edge, $e_{max} = (a, v_{max})$, first. (Pseudo-)coalescing a and v_{max} is optimal for the clique ray.

Fig. 4(b) shows a *path*, two interfering nodes u and v, and a distinct path between them connected only by φ-edges. Here, we assume that there are no interference edges between nodes on the path, other than u and v, and that there is only one path between u and v via φ-edges. Clearly, the sets $S(u)$ and $S(v)$ must be distinct, so exactly one φ-edge on this path cannot be coalesced. The optimal edge, in this case, will be the edge of minimum weight, the edges visited last by the DSO heuristic.

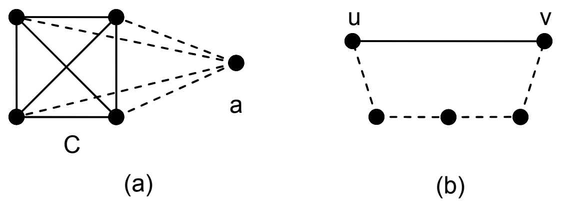


Figure 4. A 4-node clique ray (a) and a path (b).

In practice, clique rays and paths will be connected to other nodes via φ-edges. In this case, processing φ-edges in DSO order may not yield an optimal solution to aggressive (pseudo-)coalescing. It is also important to note that in the context of OCC or optimistic coalescing, optimal to the (pseudo-)coalescing does not guarantee that the overall register assignment will be optimal.

4.4 Optimistic Chordal Color Assignment

Similar to chordal coloring, vertices are processed in PEO order. Now, suppose that vertex v is the first vertex in $S(v)$ to be assigned a color, c. Optimistically, we assign color c to every other vertex in $S(v)$ as well. This may lead to a temporarily illegal coloring, but a legal color for some other vertex $u \in S(v)$ can always be found when u is processed later in the PEO. Different techniques for color assignment are applied to each subsequent vertex in the PEO, depending on whether or not it is incident on any φ-edges.

Suppose that v_i is not incident on any φ-edges. Since G is a chordal graph, we know that there will be at least one color available for v_i that is not assigned to a vertex in $N_i(v_i)$; however, we may have optimistically assigned color c to some vertex $u \in N(v_i) - N_i(v_i)$. If possible, we should attempt to respect this optimistic color assignment; if the same color is assigned to v_i, then u will have to change its color when u is processed later in the PEO. First, we look for an available color that has not been assigned to any vertex in $N(v_i)$. If one such color is available, it is assigned to v_i. If one such color is not available, then we resort to traditional chordal color assignment for v_i, and a color not assigned to a vertex in $N_i(v_i)$ is assigned to v_i. In short, the heuristic makes the effort to respect the optimistic assignment of colors to vertices later in the PEO, but will sacrifice the optimistic assignment in order to find a legal color for v_i.

Now, suppose that v_i is not the first vertex in $S(v_i)$ to be assigned a color. Thus, v_i has already been assigned some color c optimistically. If no vertex in $N_i(v_i)$ has been assigned color c, then v_i retains the color—e.g. optimistic color assignment was successful. On the other hand, if c is not available, then some other color must be assigned to v_i. In this case, v_i is removed from $S(v_i)$, and a new color is selected. This process is rather complex.

First and foremost, v_i considers incident φ-edges (v_i, x), where $x \notin S(v_i)$. If possible, v_i can be added to $S(x)$, which will eliminate some different copies involving v_i. Clearly, v_i cannot interfere with any vertex in $S(x)$. Second, if color c has been optimistically assigned to vertices in $S(x)$, then no vertex in $N_i(v_i)$ can be assigned color c. If both of these criteria are satisfied, then v_i can be added to $S(x)$.

If no color has yet been optimistically assigned to $S(x)$, then the color assigned to v will be optimistically propagated to the vertices in $S(x)$. Ideally, one would look for a color not assigned to any vertices in either $N_i(v_i)$ or $N(S(x))$; however, if no such color is found, the color that does the least amount of damage to $S(x)$ should be chosen. It may be possible that the best choice is *not* to add v_i to $S(x)$. There may be multiple sets $S(x)$ to which v_i can be added. For each set, the best color is chosen to assign to v_i, and the number of dynamically executed copies that would be eliminated by adding v_i to each set is estimated. Among all sets, the best one is then chosen for v_i.

In conclusion, the optimistic chordal coloring heuristic may assign a color to a vertex before it is processed in the PEO. The heuristic makes every attempt to respect the optimistic assignment, but will undo the assignment in order to avoid an illegal coloring. The final decision regarding the color to assign to each vertex is still made in PEO order. The final assignment of colors to vertices is never undone.

4.5 Refinement

OCC is a heuristic. The sets of pseudo-coalesced vertices that are formed early in the color assignment may be broken up early due to colors assigned to interfering vertices. After color assignment, we make one last past over the set of φ-edges. Let $e = (u, v)$ be one such φ-edge, and assume that $f(u) \neq f(v)$. Without loss of generality, if $f(v)$ is not assigned to any neighbor of u, and swapping u's color to $f(v)$ will reduce the number of dynamically executed copies, then there is no reason *not* to make the swap. Likewise, an analogous decision can be made for v. If both colors are available for both vertices, the choice that leads to a greater reduction in the number of dynamically executed copies is chosen. The effectiveness of this *refinement* step is marginal; however, there is no reason not to apply it when useful.

5. EXPERIMENTAL RESULTS

Here, we compare the results of optimistic chordal coloring to iterated register coalescing (IRC) [12] and the optimal ILP formulation of the register coalescing problem by Grund and Hack [13]. We found that both OCC and IRC produce optimal or near-optimal solutions in many cases, and that optimal chordal coloring runs faster than iterated coalescing in most cases. The two heuristics were implemented in C++, while the ILP was solved using CPLEX. The code that generates the ILP from the input graph description was written in Java, and is thus interpreted rather than compiled.

17 applications were selected from the Mediabench [20] and MiBench [14] suites and compiled and profiled using Machine SUIF [25]. Following profiling, each procedure in each application was converted to SSA Form and its interference graph was constructed. The reported runtimes are for the color assignment heuristics only, and do not reflect the costs of liveness analysis and interference graph construction.

Table 2 lists each of the benchmarks and the number of registers used in the respective experiment. For each benchmark, the number of registers allocated was the maximum chromatic number among the interference graphs of each procedure. Since SSA Form was used and the interference graphs are chordal, the chromatic number can be computed optimally in polynomial time. The number of registers used for each procedure is thus the smallest number of registers that can ensure that no variables are spilled during register assignment.

Table 2. Number of registers allocated to an ASIP for each benchmark

Benchmark	Registers	Benchmark	Registers
adpcm_coder	14	jpeg_cjpeg	18
adpcm_decoder	14	jpeg_djpeg	39
blowfish	14	mpeg2dec	21
crc32	8	mpeg2enc	45
dijkstra	6	patricia	9
FFT	13	pegwit	13
g721_decoder	16	sha	10
g721_encoder	16	susan	20
gsm	16		

Tables 3 and 4 list the experimental results. Table 3 lists the number of copies dynamically executed for each register assignment heuristic/algorithm; data is not listed for IRC/OCC when the result is the same as the optimal solution. Table 4 lists the runtime (normalized to OCC).

From Table 3, we can see that IRC and OCC found solutions that were optimal in all but three benchmarks. For *gsm* and *jpeg_djpeg*, both IRC and OCC found the same solution; for *pegwit*, OCC found a better solution than IRC, and it is only 1 copy away from the optimal. The difference in solution quality between the heuristics and optimal was 2990 dynamic copies for *gsm* and 6283 for *jpeg_djpeg*. The averages are dominated by the 4 largest benchmarks.

In all of these experiments, the translation out of SSA Form was performed using swap instructions. The number of dynamically executed copies may increase (along with possibly requiring 1 additional register) if only copies are used due to the swap problem. On the other hand, if *k*-wide parallel copies are supported, the number of dynamic copies may be reduced as well, depending on how many can be issued in parallel.

Table 4 shows the runtimes of optimal and IRC normalized to OCC. The optimal heuristic was run on a *Dell Latitude D420* laptop with an *Intel Core Duo U2500* processor running at 1.2 GHz and with 2.0 GB of RAM; the operating system was Ubuntu Feisty, and CPLEX 7.0 was used to solve the ILP. Although CPLEX 7.0 supports multi-core processing, a license supporting this feature was unavailable at the time the experiments were performed. Thus only one of the two available cores was utilized. The IRC and OCC heuristics were run on a *Dell Latitude D810* laptop with an *Intel Pentium M* processor running at 2.0 GHz with 1.0 GB of RAM; the operating system used was Fedora Core.

From Table 4, it is easy to see that the runtime of the optimal solution runs is at least one order of magnitude greater than that of OCC. In all but 3 benchmarks, IRC ran slower than OCC; however, for *crc32*, *dijkstra*, and *fft*, IRC was faster. These benchmarks contained only a few relatively simple graphs that were among the fastest to color for all 3 heuristics. IRC, whose color assignment phase is based on simplification, colored these graphs quickly. OCC suffered runtime overhead due to the cost of computing a PEO in addition to simplification, as well as the refinement phase, which did not change the solution.

5.1 The Effectiveness of Simplification

We re-ran the experiments for OCC with the simplification phase disabled. The solutions produced for three benchmarks changed as a result. For *gsm*, the number of dynamically executed copies increased from 1912177 to 1938818, an increase of 1.01x; for *pegwit*, the number of dynamically executed copies increased from 49 to 50; and for *susan*, the number of dynamically executed copies increased from 2 to 138.

Proportionally, the increase in dynamically executed copies was significant for *susan*, but in reality, a few hundred clock cycles is not particularly troubling. Nonetheless, the impact of disabling simplification may potentially be much greater for different and larger benchmarks.

Table 3. Number of dynamically executed copy instructions for each benchmark

	Dynamically Executed Copies		
Benchmark	Optimal	IRC	OCC
adpcm_rawcaudio	6995016	-	-
adpcm_rawdaudio	6995016	-	-
blowfish	0	-	-
crc32	53322406	-	-
dijkstra	0	-	-
fft	8209	-	-
g721_decode	0	-	-
g721_encode	0	-	-
gsm	1909187	1912177	1912177
jpeg_cjpeg	541326	-	-
jpeg_djpeg	272636	278919	278919
mpeg2dec	2115	-	-
mpeg2enc	95197	-	-
patricia	1820	-	-
pegwit	48	72	49
sha	442	-	-
susan	2	-	-
Average	4126084	4126630	4126629

Table 4. Runtime of the optimal and IRC heuristics, normalized to the runtime of OCC

	Runtime (Normalized to OCC)	
Benchmark	Optimal	IRC
adpcm_rawcaudio	64.15	2.13
adpcm_rawdaudio	158.32	1.98
blowfish	96.97	1.28
crc32	1.50	0.18
dijkstra	8.24	0.84
fft	6.84	0.83
g721_decode	85.35	1.50
g721_encode	113.20	1.52
gsm	56.82	1.81
jpeg_cjpeg	84.96	1.65
jpeg_djpeg	684.51	2.17
mpeg2dec	91.97	1.61
mpeg2enc	97.52	1.75
patricia	20.84	3.27
pegwit	7851.58	2.14
sha	24.75	3.13
susan	11415.79	2.47
Average	2339.33	1.89

5.2 The Effectiveness of BMCS

Next, we replaced the BMCS with the traditional MCS and computed the PEO. A minor speedup was observed due to the fact that BMCS entails a small amount of sorting of the lists required to implement PEO construction.

The quality of the solution was degraded for three benchmarks. For *gsm*, the number of dynamically executed copies increased from 1912177 to 2739189, an increase of 1.43x; for *mpeg2enc*, the number of dynamically executed copies increased from the optimal value of 95197 to 116669, an increase of 1.23x. For *pegwit*, the number of copies increased from 49 to 71.

When simplification was disabled in conjunction with using an MCS, the results were considerably worse for many of the benchmarks. In practice, there is hardly any reason why a BMCS should not be used instead of an MCS; the runtimes of the other phases of allocation dominate the difference.

Overall, the effect of simplification is greatest when many vertices incident on high-weight φ-edges are placed toward the end of the PEO. Removing interfering neighbors that occur earlier in the PEO significantly reduces the constraints on these vertices when their colors are assigned. At the same time, BMCS makes an extra effort to ensure that the very same vertices would be placed as early as possible in the PEO; and thus, simplification tends to remove vertices that are not incident on high-weight φ-edges that occur later in the PEO. Thus, the overall effect and effectiveness of these optimizations appear to be quite similar.

5.3 The Effectiveness of Refinement

Lastly, we re-ran the experiments for OCC with only refinement disabled. The number of dynamically executed copies increased for 4 benchmarks: for *gsm* it increased from 1912177 to 1938818, the same result as when simplification was disabled in Section 6.1; for *jpeg_djpeg* and *pegwit*, 1 additional copy was executed dynamically; and for *mpeg2enc*, the number increased from 95197 to 96150, an increase of 1.01x. With the exception of *gsm*, refinement proved to be relatively ineffective on its own, and if simplification is used, it becomes even less effective. Nonetheless, only a handful of φ-edges appear to lead to the introduction of copies, so the overhead of refinement is limited.

If the other phases of the algorithm do a good job with respect to color assignment, a post-processing phase, such as refinement, will have minimal opportunity to improve the quality of the solution. On the other hand, unless compile-time is of primary concern, there is practically no harm in running refinement following an initial color assignment.

6. CONCLUSION

A novel register allocation technique for chordal graphs that is both optimistic and conservative has been proposed in this paper. This technique, called optimistic chordal coloring (OCC), does not employ coalescing; doing so would destroy the chordal property of interference graphs for SSA Form programs. By ensuring that the interference graph remains chordal, and that colors are assigned to vertices in PEO order, OCC ensures that the chromatic number of the interference graph is never increased.

Similar in principle to the optimistic coalescing heuristic of Park and Moon [22], OCC employs an aggressive pseudo-coalescing phase to identify independent sets of variables that should receive the same color under ideal circumstances. OCC makes every effort to assign these vertices the same color, and only deviates from this scheme when the desired color is not available.

In practice, both OCC and iterated register coalescing produce similar—often optimal—results on a set of chordal interference graphs generated from a set of embedded applications. At present, it is not clear whether these graphs represent coincidentally easy problem instances, or whether copy folding during SSA Form effectively eliminates so many copies that the resulting interference graph is pragmatically easier to color well.

In the future, we intend to test the OCC heuristic on chordal interference graphs generated from a larger set of applications, including those from outside of the embedded domain.

ACKNOWLEDGMENT

We wish to thank Sebastian Hack for providing the optimal solutions reported in Section 5. Florent Bouchez helped to identify a bug in our implementation of iterated register coalescing, which significantly skewed the results in the initial draft of this paper; his contribution led us to completely redesign the algorithms to become competitive with iterated register coalescing. Florent Bouchez, Alain Darte, Sebastian Hack, and Fabrice Rastello provided insightful comments on the final version of this paper, and participated in numerous discussions that helped improve its overall quality.

REFERENCES

[1] Bouchez, F., Darte, A., Guillon, C., and Rastello, F. *Register Allocation and Spill Complexity Under SSA*, Technical Report 2005-33, ENS-Lyon, Lyon France, 2005.

[2] Bouchez, F., Darte, A., Guillon, C., and Rastello, F. Register allocation : what does the NP-Completeness proof of Chaitin et al. really prove? Or revisting register allocation: why and how. *In Proc. of the 19th International Workshop on Languages and Compilers for Parallel Computing (LCPC '06)*, (New Orleans, LA, USA, November 2-4, 2006)

[3] Bouchez, F., Darte, A., and Rastello, F. On the complexity of register coalescing. *In Proc. of the International Symposium on Code Generation and Optimization (CGO '07)* (San Jose, CA, USA, March 11-14, 2007) 102-114

[4] Briggs, P., Cooper, K. D., Harvey, T. J., and Simpson, L. T. Practical improvements to the construction and destruction of static single assignment form. *Software—Practice and Experience vol. 28, no. 8,* July, 1998, 859-881.

[5] Briggs, P. Cooper, K. D., and Torczon, L. Improvements to graph coloring register allocation. *ACM Trans. Programming Languages and Systems, vol. 16, no. 3*, May, 1994, 428-455.

[6] Brisk, P., Dabiri, F., Jafari, R., and Sarrafzadeh, M. Optimal register sharing for high-level synthesis of SSA form programs. *IEEE Trans. Computer Aided Design, vol. 25, no. 25*, May, 2006, 772-779.

[7] Budimlić, Z., Cooper, K. D., Harvey, T. J., Kennedy, K., Oberg, T. S., and Reeves, S. W. Fast copy coalescing and live range identification. *In Proc. of the International Conf. on Programming Language Design and Implementation (PDLI '02)*, (Berlin, Germany, June 17-19, 1992) 25-32.

[8] Chaitin, G. J. Register allocation and spilling via graph coloring. *In Proc. of the 1982 SIGPLAN Symp. on Compiler Construction*, (Boston, MA, USA, June 23-25, 1982), pp. 98-101.

[9] Choi, J-D., Cytron, R., and Ferrante, J. Automatic construction of sparse data flow evaluation graphs. *In Proc. of 18th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL '91)* (Orlando, FL, USA, Jan. 21-23, 1991) 55-66.

[10] Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N., and Zadeck, F. K. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Programming Languages and Systems, vol. 13, no. 4,* October, 1991, 451-490.

[11] Gavril, F. Algorithms for minimum coloring, maximum clique, minimum coloring by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput., vol. 1, no. 2*, June, 1972, 180-187.

[12] George, L., and Appel, A. Iterated register coalescing. *ACM Trans. Programming Languages and Systems, vol. 18, no. 3,* May, 1996, 300-324.

[13] Grund, D., and Hack, S. A fast cutting-plane algorithm for optimal coalescing. *In Proc. of the 16th International Conference on Compiler Construction (CC '07)* (Lisbon, Portugal, March 26-27, 2007) 111-125.

[14] Guthaus, M. R., et. al. . MiBench: a free commercially representative embedded benchmark suite. *In Proc. of the 4th Annual IEEE Workshop on Workload Characterization (WWC '01)* (Austin, TX, USA, December 2, 2001) 3-14.

[15] Hack, S., and Goos, G. Optimal register allocation for SSA-form programs in polynomial time. *Information Processing Letters vol. 98, nol 4*, May, 2006, 150-155.

[16] Hack, S., Grund, D., and Goos, G. Register allocation for programs in SSA Form, *In Proc. of the 15th International Conference on Compiler Construction (CC '06)* (Vienna, Austria, March 30-31, 2006) 247-262.

[17] Hailperin, M. Comparing conservative coalescing criteria. *ACM Trans. Programming Languages and Systems, vol. 27, no. 3,* May, 2005, 571-582.

[18] Kaluskar, V. P. *An Aggressive Live Range Splitting and Coalescing Framework for Efficient Register Allocation*. M.S. Thesis, Georgia Institute of Technology, November, 2003.

[19] Kempe, A. B. On the geographical problem of the four colors. *American Journal of Mathematics*, *vol.* 2*,* 1879, 193-200.

[20] Lee, C., Potkonjak, M., and Mangione-Smith, W. H. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. *In Proc. of the 30th International Symposium on Microarchitecture (MICRO-30, '97)* (Research Triangle Park, NC, USA, December 1-3, 1997) 330-335.

[21] Matula, D. W., and Beck, L. L. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM, vol. 3, no. 3*, July, 1983, 417-427.

[22] Park, J., and Moon, S-M. Optimistic register coalescing, *ACM Trans. Programming Languages and Systems, vol. 26, no. 4*, July, 2004, 735-765.

[23] Poletto, M., and Sarkar, V. Linear scan register allocation. *ACM Transactions on Programming Languages and Systems, vol. 21, no. 5*, September, 1999, 895-913.

[24] Rastello, F., de Ferriere, F., and Guillon, C. Optimizing translation out of SSA using renaming constraints. *In Proc. of the 2nd International Symp. on Code Generation and Optimization (CGO '04)* (Palo Alto, CA, USA, March 21-24, 2004) 265-278

[25] Smith, M. D., and Holloway, G. An introduction to Machine SUIF and its portable libraries for analysis and optimization. *Technical Report*. Harvard University. July 15, 2002.

[26] Sreedhar, V. C., Ju, R. D-C., Gillies, D. M., and Santhanam, V. Translating out of static single assignment form. *In Proc. of the 6th International Symposium on Static Analysis (SAS '99)*, (September 22-24, Venezia, Italy, 1999) 194-210

[27] Tarjan, R. E., and Yannakakis, M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput. vol. 13, no. 3*, August, 1984, 566-579.

[28] Vegdahl, S. R. Using node merging to enhance graph coloring. *In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99)* (Atlanta, GA, USA, May 1-4, 1999) 150-154.