Karlsruhe Institute of Technology

Institut für Programmstrukturen
und Datenorganisation (IPD)

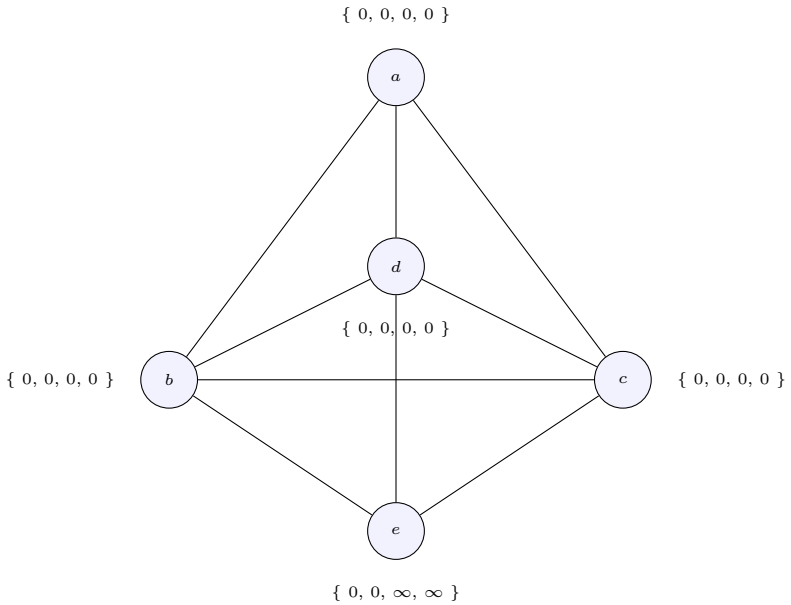
Lehrstuhl Prof. Dr.-Ing. Snelting

SSA-basierte Registerzuteilung mit integrierter Kopienminimierung

Studienarbeit von

Thomas Bersch

an der Fakultät für Informatik



Gutachter: Prof. Dr.-Ing. Gregor Snelting

Betreuender Mitarbeiter: Dipl.-Inform. Sebastian Buchwald

Bearbeitungszeit: 16. Dezember 2009 – 14. Juni 2010

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 14. Juni 2010

Kurzfassung

Registerzuteilung ist eine wichtige Aufgabe während des Übersetzungsvorgangs von Programmen. Eine Registerzuteilung kann großen Einfluss auf die Laufzeit der übersetzten Programme haben. Besonders für SSA-basierte Zwischensprachen ist das Vermeiden unnötiger Registerkopien ein weiterer wichtiger Bestandteil.

Mit dem PBQP (Partitioned Boolean Quadratic Problem) lassen sich beide Aufgaben zusammenfassen und als gemeinsames Optimierungsproblem lösen. Diese Arbeit beschäftigt sich damit, wie eine Abbildung dieser Aufgaben auf PBQP erfolgen kann und wie gut sich das Verfahren im Hinblick auf die Qualität der Lösung und die nötige Übersetzungszeit eignet. Da das Finden einer gültigen PBQP-Lösung ein NP-vollständiges Problem ist, wird in dieser Arbeit auf eine Heuristik zurückgegriffen, mit der die Laufzeit linear in Programmgröße. Dazu wird auch untersucht, wie garantiert werden kann, dass eine gültige Lösung gefunden wird.

Inhaltsverzeichnis

1 Einführung

In heutigen Compilern unterscheidet man zum einen das Frontend, welches sich um die Analyse und Transformation eines Programms in eine Zwischensprache kümmert, und das Backend, das aus dem Zwischensprachenprogramm anschließend ein Programm in der gewünschten Zielsprache erzeugt. Eine wichtige Aufgabe des Backends ist dabei die Registerzuteilung, deren Ziel es ist, die in der Zwischensprache verwendeten symbolischen Register möglichst optimal auf die real vorhandenen Register der gewünschten Zielarchitektur abzubilden. Hierbei muss auch berücksichtigt werden, dass für komplexere Berechnungen die Anzahl der Register möglicherweise nicht ausreicht und Werte in den Speicher ausgelagert werden müssen. Da Speicherzugriffe im Vergleich zu Registerzugriffen deutlich langsamer sind, wird für eine optimale Lösung versucht, diese Zugriffe möglichst zu minimieren.

Ein häufig verwendeter Ansatz zur Registerzuteilung ist die Abstraktion auf das Problem der Graphfärbung [Cha04, BCT94]. Analog zur Zuteilung von Registern wird hier versucht einen Graphen mit möglichst wenigen Farben so zu färben, dass zwei Knoten die gleichzeitig lebendig sind, nie dieselbe Farbe zugeteilt wird. Diese Aufgabe ist ein NP-vollständiges Problem. Inzwischen gibt es aber Verfahren, die eine optimale Färbung in annähernd lineare Zeit ermöglichen.

Neben der eigentlichen Registerzuteilung wird speziell für SSA-basierte Zwischensprachen die Kopienminimierung immer wichtiger. Diese versucht unnötige Registerkopien im Programm zu finden und wenn möglich zu entfernen.

Für beide Aufgaben bietet die SSA-Form besondere Eigenschaften, die verwendet werden können. So sind die Interferenzgraphen von Programmen in SSA-Form chordal und es existiert eine Reihenfolge, in der die Registerzuteilung immer gelingt. Dies ermöglicht den Registerbedarf schon vor der Registerzuteilung zu bestimmen, was bei anderen Zwischensprachen in der Regel nicht möglich ist.

In dieser Arbeit wird nun im Zusammenhang mit SSA-basierter Registerzuteilung ein Verfahren untersucht werden, welches schon zuvor erfolgreich zur Registerzuteilung für nicht SSA-basierte Zwischensprachen verwendet wurde. Das Verfahren basiert auf dem Partitioned Boolean Quadratic Problem (PBQP), dass wie das Integer Linear Programm (ILP) zur Gruppe der mathematischen Optimierungsprobleme gehört. Neben der eigentlichen Registerzuteilung ermöglicht dieses Verfahren auch eine gleichzeitige Kopienminimierung. Dazu werden wir den ersten Ansatz zur Registerzuteilung um eben diese Möglichkeit erweitern und zeigen, dass das Verfahren immer eine gültige Zuteilung findet. Weiterhin werden wir einen leicht modifizierten Ansatz des Verfahrens zur Lösung von PBQP vorstellen und untersuchen, ob dadurch eine noch bessere Registerzuteilung erreicht werden kann.

2 Grundlagen

2.1 SSA-Form – Static Single Assignment Form

Die Static Single Assignment-Form (SSA-Form) ist eine spezielle Form der Zwischensprachendarstellung. Ein Programm ist in SSA-Form, wenn an jede Variable an genau einer Stelle im Programm eine Zuweisung erfolgt. Wird einer Variable mehrmals ein Wert zugewiesen, so muss diese bei der Transformation in so viele Variablen aufgeteilt werden wie Zuweisungen vorhanden sind. Üblicherweise wird der ursprüngliche Variablenname und eine Versionsnummer als Index verwendet. Bei der Transformation müssen dann alle Verwendungen durch die entsprechende Version der letzten Zuweisung ersetzt werden (siehe Abbildung 2.1). Innerhalb von Grundblöcken ist dies einfach, da die vorangegangene Definition eindeutig ist. Ist die letzte Zuweisung aber vom Ablaufpfad des Programmes abhängig, so ist zunächst nicht klar welche Version der Variable verwendet werden muss. Um dieses Problem zu lösen verwendet man eine spezielle Notation, die ϕ-Funktion. Dabei handelt es sich um eine Auswahlfunktion ϕ, die aus der Menge der dem Programmablauf entsprechenden Vorgängerdefinitionen die letzte verwendete Definition var_i auswählt und daraus eine neue Definition erzeugt (siehe auch [RWZ88]). Betrachten wir hierzu Abbildung 2.2: In den beiden Grundblöcken GB_1 und GB_2 wird jeweils eine Version x_i der Variablen x definiert. Danach laufen die beiden Grundblöcke in Grundblock GB_3 zusammen. Dies kann z.B. durch eine Verzweigung oder Schleife hervorgerufen sein. Zu Beginn von GB_3 ist nun unklar welcher Grundblock zuletzt ausgeführt wurde und welcher Wert somit der aktuell gültige ist. Die ϕ-Funktion weist nun, je nachdem welcher Grundblock zuletzt ausgeführt wurde entweder x_1 oder x_2 der neuen Variable x_3 zu.

```
1 x = 5;                    1 x1 = 5;
2 print(x);        =>       2 print(x1);
3 x = 10;                   3 x2 = 10;
```

Abbildung 2.1: Transformation in SSA-Form

2.2 Firm und libFirm

Firm ist eine graphbasierte Zwischensprache, die die SSA-Form implementiert. Entwickelt wurde Firm an der Universität Karlsruhe (TH) [TLB99]. Firm stellt ein

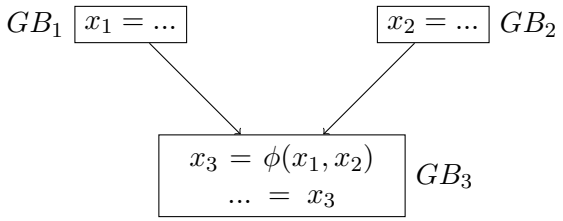


Abbildung 2.2: Variablenauswahl mittels ϕ-Funktion

Programm als gerichteten Graphen dar, in dem Datenabhängigkeiten und Steuerfluss kombiniert sind. Die Knoten des Graphen repräsentiern den aus der jeweiligen Operation resultierenden Wert. Die Datenabhängigkeitskanten zeigen auf die zur Berechnung des Ergebnisses benötigten Argumente. Die Reihenfolge der Befehle innerhalb eines Grundblocks ist nicht explizit vorgegeben, sondern ergibt sich aus den Datenabhängigkeiten. So kann beispielsweise eine Multiplikation erst durchgeführt werden, wenn alle benötigten Argumente zur Verfügung stehen. Für Grundblöcke hingegen wird eine Reihenfolge teilweise durch den Steuerfluss gestgelegt.

libFirm [Lin02] ist die Implementierung von Firm in Form einer C-Bibliothek.

2.3 Perfektes Eliminationsschema und chordale Graphen

Ein perfektes Eliminationsschma (PES) ist ein Begriff aus der Graphentheorie. Es gibt eine Reihenfolge vor, in der Knoten aus einem Graphen entfernt werden müssen, so dass für jeden entfernten Knoten v gilt: alle Nachbarn von v bilden eine Clique. Es gilt außerdem der nachfolgende Satz:

Satz 1 *Ein Graph ist genau dann chordal, wenn er ein perfektes Eliminationsschema besitzt.*

Ein Graph G heißt chordal, wenn er keinen Zyklus als Untergraphen enthält, der länge als drei ist. Chordale Graphen sind perfekt, d.h die minimale Anzahl der benötigten Farben $\chi(G)$ für ein Färbung entspricht der Anzahl der Knoten $\omega(G)$ der größten Clique.

Berechnen lässt sich ein perfektes Eliminiationsschema in polynomieller Zeit. Für die Registerzuteilung heißt das, dass sich mittels eines perfekten Eliminationsschema eine gültige Registerzuteilung in polynomieller Zeit finden lässt.

2.4 Registerzuteilung

2.4.1 Allgemein

Die Registerzuteilung ist eine Phase während des Übersetzungsvorgangs. Sie versucht allen symbolischen Registern eines Programms ein reales Register der entsprechenden Zielarchitektur zuzuweisen. In der Regel ist die Zahl der symbolischen Register deutlich größer als die vorhandenen realen Register einer CPU. Für eine Zuteilung mit möglichst wenigen Registern ist dies ein NP-vollständiges Problem.

Ein oft verwendeter Ansatz zur Registerzuteilung ist die Abstraktion auf das Problem der Graphfärbung [Cha04, BCT94]. Die Grundlage dazu bildet der sogenannte Interferenzgraph. Die Knoten eines Interferenzgraphen entsprechen den symbolischen Registern, denen eine Farbe bzw. ein Register zugeteilt werden soll und die Kanten entsprechen Konflikten (Interferenzen) zwischen je zwei symbolischen Registern, die entstehen wenn diese gleichzeitig lebendig sind. Das Ziel der Graphfärbung ist es nun den Interferenzgraphen so zu färben, dass zwei interferierenden Knoten nie dieselbe Farbe zugeteilt wird. Auf Registerebene bedeutet dies, dass zwei gleichzeitig lebendige Werte nie dasselbe Register erhalten.

2.4.2 SSA-basierte Registerzuteilung

Die SSA-Form bietet für die Registerzuteilung einige Vorteile gegenüber anderen, nicht SSA-basierten Zwischensprachen. So sind die Interferenzgraphen von Programme die sich in dieser Form befinden immer chordal [Hac05, BDR07], weshalb es für diese Interferenzgraphen auch immer ein perfektes Eliminationsschema (siehe Abschnitt 2.3) gibt. Für chordale Interferenzgraphen gilt außerdem, dass die Anzahl der benötigten Register druch die Anzahl der Knoten der größten Clique beschränkt ist. Dadurch sit es möglich, die Anzahl der benötigten Register und die Programmstellen an denen nicht genügend Register vorhanden sind schon vor der Registerzuteilung zu bestimmen. Aus diesem Grund lässt sich für SSA-basierte Zwischensprachen die Auslagerungsphase von der Registerzuteilung entkoppeln. Diese stellt sicher, dass an jeder Programmstelle der Registerdruck gleich oder kleiner als die vorhandene Anzahl an Registern ist.

2.5 Kopienminimierung

Bei der Registerzuteilung kann es vorkommen, dass Operanden eines Befehls nicht in den vom Befehl vorgegebenen bzw. dem Befehl zugeteilten Registern stehen. Ist dies der Fall, so muss der Registerinhalt durch Kopie oder Permutation in dem entsprechenden Register bereitgestellt werden. Dazu müssen die nötigen Operationen in den Code eingefügt werden. Im Folgenden werden die beiden wichtigsten Ursachen für die Entstehung von solchen Kopien oder Permutationen erläutert.

2.5.1 Registerbeschränkungen

Für einige Befehle sind die Register nicht frei wählbar, sondern unterliegen Beschränkungen. Diese legen fest, welche Register für Ergebnis und Operanden verwendet werden dürfen. Unter Umständen legen sie die zu verwendenden Register sogar eindeutig fest. Können diese Beschränkungen durch die Registerzuteilung nicht direkt eingehalten werden, gibt es für das Programm zunächst keine gültige Zuteilung. Um eine solche wieder zu ermöglichen müssen nachträglich Kopien in das Programm eingefügt werden, die die Werte in den geforderten Registern bereitzustellen.

Beispiel 1 *Seien OP1 und OP2 zwei Funktionen/Befehle, die zwei Operanden miteinander verknüpfen und r0, r1 und r2 drei Register. OP1 sei unbeschränkt und für OP2 sei nur der erste Operand auf das Register r0 festgelegt. Das Zielregister, sowie das Register des zweiten Operanden seinen frei wählbar.*

Betrachten wir nun die beiden Codestücke aus Listing 2.1 und Listing 2.2. Der Wert von r1 soll mit dem Ergebnis von OP1(r0, r1) verknüpft werden. In Listing 2.1 wurden die Register ungünstig gewählt. Aufgrund der Beschränkung von OP1 muss das Ergebnis von r2 in r0 kopiert werden. Verwendet man wie in Listing 2.2 für das Ergebnis aus OP1 das Register r0, so steht der Wert schon in dem von OP2 erwarteten Register und die Kopie kann eingespart werden.

```
1 r2 = OP1(r0, r1);
2 r0 = r2;
3 r0 = OP2(r1);
4 ...
```

Listing 2.1: Schlechte Registerwahl

```
1 r0 = OP1(r0, r1);
2 r0 = OP2(r1);
3 ...
4
```

Listing 2.2: Gute Registerwahl

Kopien können aber auch zwingend erforderlich sein. Schreibt beispielsweise ein Befehl das Ergebnis seiner Berechnung immer in dasselbe Register, so muss das Register vor der nächsten Ausführung des Befehls gesichert werden, sofern die Ergebnisse interferieren.

2.5.2 SSA-Abbau

Ein weiterer Grund, wodurch Kopien entstehen können ist der SSA-Abbau, genauer die Auflösung von ϕ-Funktionen. Da es für diese Funktionen keine Hardwareunterstützung gibt, muss die Semantik der Funktion nachgebildet werden. Die ϕ-Funktion sorgt dafür, dass im aktuellen Grundblock der dem Steuerfluss entsprechende Wert verwendet wird. Hierzu legt sie den Wert der Variable aus dem im Steuerfluss vorangegangenen Grundblock in einer neuen Variable ab. Auf Registerebene bedeutet dies, dass der zu verwendende Wert beim Eintritt in den Grundblock in einem bestimmten Register verfügbar ist. Dies kann auf zwei Arten erreicht werden. Entweder

schreibt der berechnende Befehl aus dem Vorgängergrundblock den Wert direkt in das geforderte Register oder der Wert muss vor dem Verlassen des Vorgängergrundblocks in das Register kopiert werden, wodurch eine Registerkopie entsteht. Ziel ist es also, die Register so zuzuteilen, dass die benötigten Ergebnisse direkt in die erwarteten Register geschrieben werden und nicht erst noch kopiert werden müssen.

2.6 PBQP - Partitioned Boolean Quadratic Problem

Das Partitioned Boolean Quadratic Problem (PBQP) [SE02] gehört als Sonderfall des Quadratic Assignment Problem (QAP) zu den Optimierungsproblemen und wurde schon erfolgreich zur Befehlsauswahl [EKS03] und Registerzuteilung eingesetzt [HS06]. Das Ziel ist es, aus einer Menge von nicht notwendigerweise unabhängigen Auswahlmöglichkeiten jeweils die Alternative auszuwählen, für die die Gesamtkosten der Lösung minimal werden. Für die Berechnung einer gültigen Lösung ist PBQP NP-vollständig. In einigen Fällen lässt sich jedoch durch die Verwendung einer Heuristik eine Lösung finden, für die die Laufzeit linear in der Anzahl der Knoten ist. Findet man mittels der Heuristik keine Auswahl, sodass die Lösung gültig ist, so sind die Kosten der getroffenen Auswahl unendlich.

2.6.1 Das Partitioned Boolean Quadratic Problem

Sei $n \in \mathbb{N}$ die Anzahl der Auswählmöglichkeiten einer PBQP-Instanz und $i, j \in \{1, \ldots, n\}$. Jede der n Auswahlmöglichkeiten kann als diskrete Variable $\vec{x_i}$ mit Elementen aus 0 oder 1 angesehen werden, für die gilt, dass genau einem der Element aus $\vec{x_i}$ die 1 zugeordnet ist. Zu jeder diskreten Variable $\vec{x_i}$ gehört weiterhin eine Anzahl $d \in \mathbb{N}$ an Alternativen, sowie ein Kostenvektor $\vec{c_i} \in \{\mathbb{R} \cup \infty\}^d$, der jeder der d möglichen Alternativen Kosten zuordnet. Außerdem werden Kosten für Abhängigkeiten zwischen zwei Auswahlmöglichkeiten $\vec{x_i}$ und $\vec{x_j}$ in Form einer Kostematrix $C_{ij} \in \{\mathbb{R} \cup \infty\}^{d_i \times d_j}$ definiert. Diese ordnet jeder Kombination von Alternativen aus $\vec{x_i}$ und $\vec{x_j}$ die hierfür anfallenden Kosten zu. Eine Lösung entspricht der Auswahl einer Alternative für jede Auswahlmöglichkeit $\vec{x_i}$. Ziel ist es, die für die Menge der gewählten Alternativen anfallenden Gesamtkosten zu minimieren.

Etwas Formaler bedeutet dies:

Die Funktion

$$f = \sum_{1 \leq i \leq n} \vec{x_i}^T \vec{c_i} + \sum_{1 \leq i \leq j \leq n} \vec{x_i}^T C_{ij} \vec{x_j} \tag{2.1}$$

soll unter den Nebenbedingungen

$$\vec{x_i} \in \{0,1\}^{|\vec{c_i}|} \qquad \forall 1 \leq i \leq n$$

$$\vec{x_i}^T \vec{1} = 1 \qquad \forall 1 \leq i \leq n$$

minimiert werden.

PBQP lässt sich auch als Graph $G = (V, E)$ mit einer total geordneter Knotenmenge V und einer Kantenmenge $E = \{(u, v) \mid u < v, u \in V, v \in V\}$ formuliern. Jede diskrete Variable $\vec{x_i}$ entspricht dabei einem Knoten $u \in V$ und jede Matrix einer gerichteten Kante $(u, v) \in E$. Zusätzlich ordnen wir jedem Knoten eine Menge von Alternativen $\mathcal{A}_u = \{A_1, \ldots, A_d\}$ zu, deren Kosten durch einen Kostenvektor $\vec{c_u}$ angegeben werden. Jeder Kante (u, v) wird außerdem eine Kostenmatrix C_{uv} zugeordnet, die jedem Paar aus wählbaren Alternativen $A_k \in \mathcal{A}_u$ und $A_l \in \mathcal{A}_v$ weitere Kosten zuordnet. Die Alternative A_k des Quellknoten u wählt dabei die k-te Zeile und die Alternative A_l des Zielknoten v wählt die l-te Spalte der Matrix aus.

Eine Lösung in Form einer Auswahl kann nun als Abbildung l aufgefasst werden, die jedem Knoten u eine Alternative $l(u) \in \mathcal{A}_u$ zuordnet. Die Kosten einer gewählten Alternative $A_u \in \mathcal{A}_u$ im Knoten u seien durch $c(A_u)$ und die Kosten für Abhängigkeiten zwischen den gewählten Alternativen A_u des Knoten u und A_v eines zweiten Knoten v seien durch $c(A_u, A_v)$ beschrieben. Die Gesamtkosten einer Auswahl können dann durch

$$c(l) = \sum_{u \in V} c(l(u)) + \sum_{(u,v) \in E} c(l(u), l(v)). \tag{2.2}$$

beschrieben werden. Eine Auswahl nennen wir dann eine Lösung, wenn die Gesamtkosten endlich sind.

Für PBQP reicht es aus, eine beliebige Richtung der Kanten festzuglegen. Wichtig dabei ist nur, dass der Quell- und Zielknoten einer Kante eindeutig festlegt ist. Da man diese Ausrichtung auch als Eigenschaft der Kostenmatrix auffassen kann, können wir den PBQP-Graphen im Nachfolgenden auch als ungerichteten Graphen auffassen.

2.6.2 Lösen eines PBQP

Nachfolgend soll informell ein Lösungsalgorithmus beschrieben werden, wie er auch in KAPS (Karlsruher PBQP Solver) verwendet wird. Die Lösung erfolgt dabei in drei Teilschritten:

1. Normalisierung von Kanten sowie Reduktion von Knoten durch RI-, RII- oder RN-Reduktionen, bis keine Kante mehr vorhanden ist.

2. Auswählen des Minimum für jedem Knoten.

3. Rückpropagation der durch RI- oder RII- (bzw. auch RN-) Reduktionen entfernten Knoten.

Unabhängige Kanten

Unabhängige Kanten sind Kanten, bei denen sich die Kosten $c(A_u, A_v)$ für jede Kombination aus wählbaren Alternativen $A_u \in \mathcal{A}_u$ und $A_v \in \mathcal{A}_v$ in zwei Anteile u_i und v_j zerlegen lassen, so dass $c_{ij} = u_i + v_j$ gilt. i, j entsprechen dabei den Indizes der Alternativen A_u und A_v in den Kostenvektoren $\vec{c_u}$ und $\vec{c_v}$. Die Kostenmatrix unabhängiger Kanten ist somit folgendermaßen aufgegebaut:

$$C_{uv} = \begin{pmatrix} u_1 + v_1 & ... & u_1 + v_m \\ \vdots & \ddots & \vdots \\ u_n + v_1 & ... & u_n + v_m \end{pmatrix}$$

Indem man die Vektoren $\vec{u} = (u_1, ..., u_n)$ zum Kostenvektor $\vec{c_u}$ und $\vec{v} = (v_1, ..., v_m)$ zum Kostenvektor $\vec{c_v}$ addiert, lässt sich die Kostenmatrix in die Nullmatrix überführen. Nun entstehen für keine Kombination von Alternativen A_u und A_v mehr Kosten. Die Kante ist somit unabhängig und kann entfernt werden (siehe Abbildung 2.3). Gibt es für einen Knoten nur eine wählbare Alternative, so sind alle inzidenten Kanten unabhängig und können entfernt werden.

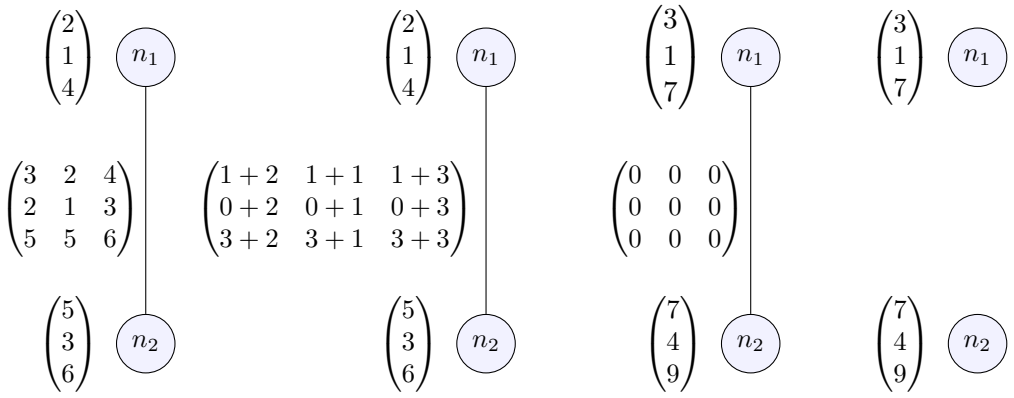


Abbildung 2.3: Entfernen unabhängiger Kanten

Normalisierung

Durch Normalisierung versucht man die Kosten einer Kante C_{uv} ganz oder zumindest teilweise in die Kostenvektoren $\vec{c_u}$ und $\vec{c_v}$ der inzidenten Knoten zu verschieben. Dies erfolgt in zwei Schritten. Zuerst wird in Richtung des ersten Vektors $\vec{c_u}$ und anschließend in Richtung des zweiten Vektors $\vec{c_v}$ normalisiert. Für eine Richtung iteriert man dazu über die Zeilen und für die andere Richtung über die Spalten der Kostenmatrix. In jedem Iterationsschritt ermittelt man das Minimum der Zeile bzw. Spalte, subtrahiert dieses Ergebnis von der kompletten Zeile oder Spalte und

addiert dann den Wert zu den Kosten der entsprechenden Alternative A_u bzw. A_v des Kostenvektors $\vec{c_u}$ oder $\vec{c_v}$. Ist die Kante unabhängig, so ist die resultierende Matrix die Nullmatrix. Eine Matrix nennt man dann in Normalform, wenn das Minimum jeder Zeile und jeder Spalte 0 ist. Ist die Matrix nicht in Normalform, so existiert eine Zerlegung

$$C = C_N + C_U$$

wobei C_N eine Matrix in Normalform und C_U die Kostenmatrix einer unabhängigen Kante ist. Die Kosten der unabhängigen Kante können in die Kostenvektoren der Knoten verschoben werden. Übrig bleibt die Kostenmatrix C_N in Normalform.

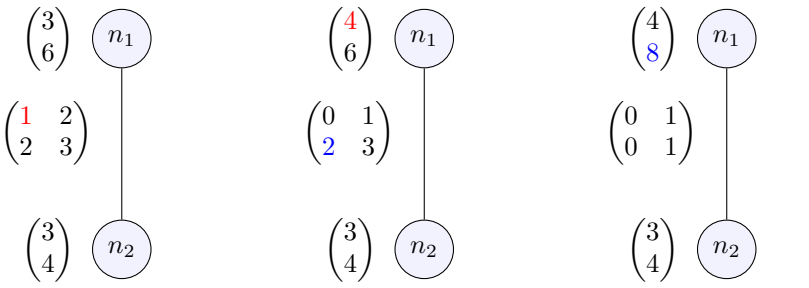


Abbildung 2.4: Normalisierung in Richtung n_1

Abbildung 2.4 zeigt die Normalisierung in Richtung des Knoten n_1. In jeder Iteration i wird das Minimum der Zeile i ermittelt und vom i-ten Zeilenvektor subtrahiert. Danach wird der Wert auf die i-te Alternative des Kostenvektors n_1 aufaddiert.

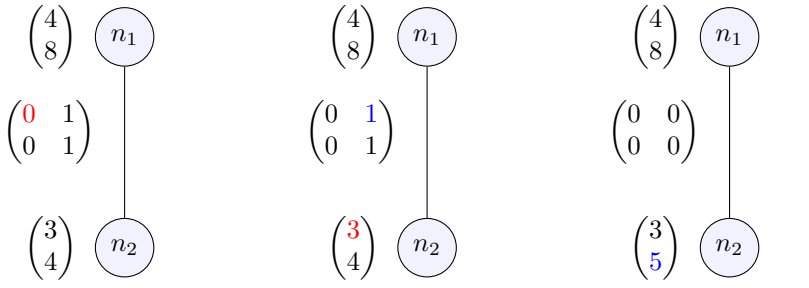


Abbildung 2.5: Normalisierung in Richtung n_2

Abbildung 2.5 zeigt die Normalisierung in Richtung des Knoten n_2. In jeder Iteration i wird das Minimum der Spalte i ermittelt und vom i-ten Spaltenvektor subtrahiert.

Danach wird der Wert auf die i-te Alternative des Kostenvektors n_2 aufaddiert. In diesem Beispiel ist die Kante sogar unabhängig und kann entfernt werden.

Triviales Lösen von Knoten mit Grad Null

Knoten vom Grad Null, also solche ohne Kanten und damit auch ohne Abhängigkeiten zu anderen Knoten können trivial gelöst werden, indem die kostengünstigste Alternative ausgewählt wird.

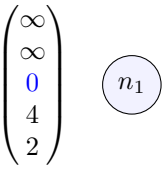


Abbildung 2.6: Knoten n_1 kann trivial gelöst werden

Knoten n_1 inAbbildung 2.6 hat keine inzidenten Kanten. Die Lösung für den Knoten kann trivial anhand des Kostenvektors ermittelt werden. In diesem Fall ist die Lösung der dritte Eintrag mit Kosten 0.

RI-Reduktionen – Entfernen von Knoten mit Grad Eins

Ein Knoten v vom Grad Eins kann, nachdem seine Kosten in den adjazenten Knoten u verschoben wurden entfernt werden. Der adjazente Knoten ist dabei nicht notwendigerweise vom Grad Eins. Hierzu werden zunächst die Kosten des Knoten v in die Kostenmatrix der Kante verschoben. Danach kann die Kante in Richtung des adjazenten Knoten u normalisiert und anschließend entfernt werden. Der Knoten wird für die spätere Rückpropagationsphase auf den Reduktionstack gelegt. Für die Berechnung des neuen Kostenvektors $\vec{c'_u}$ gilt:

$$c'(A_u) = c(A_u) + \min_{A_v \in \mathcal{A}_v} (c(A_v) + c(A_u, A_v)) \tag{2.3}$$

Abbildung 2.7 zeigt, wie ein Knoten vom Grad Eins mittels RI-Reduktion entfernt wird.

RII-Reduktionen – Entfernen von Knoten mit Grad Zwei

RII-Reduktionen sind bei Knoten v vom Grad Zwei anzuwenden. Im Gegensatz zur RI-Reduktion, bei der die Kosten des zu reduzierenden Knoten in den adjazenten Knoten verschoben werden, wird bei der RII-Reduktion eine neue Kante zwischen den beiden adjazenten Knoten u und w eingefügt und die Kosten für eine Auswahl

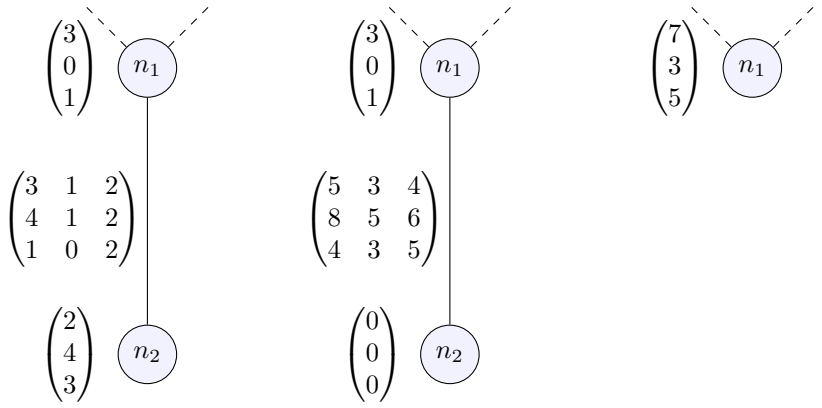


Abbildung 2.7: RI-Reduktion des Knotens n_2

im zu reduzierenden Knoten in die Kostenmatrix der neuen Kante verschoben. Für den Fall, dass es bereits eine Kante zwischen den adjazenten Knoten gibt, werden die Kosten auf die bestehende Kostenmatrix aufaddiert. Formal berechnet sich die neue Kostenmatrix $c'(A_u, A_w)$ wie folgt:

$$c'(A_u, A_w) = c(A_u, A_w) + \min_{A_v \in \mathcal{A}_v} (c(A_u, A_v) + c(A_v) + c(A_v, A_w)) \tag{2.4}$$

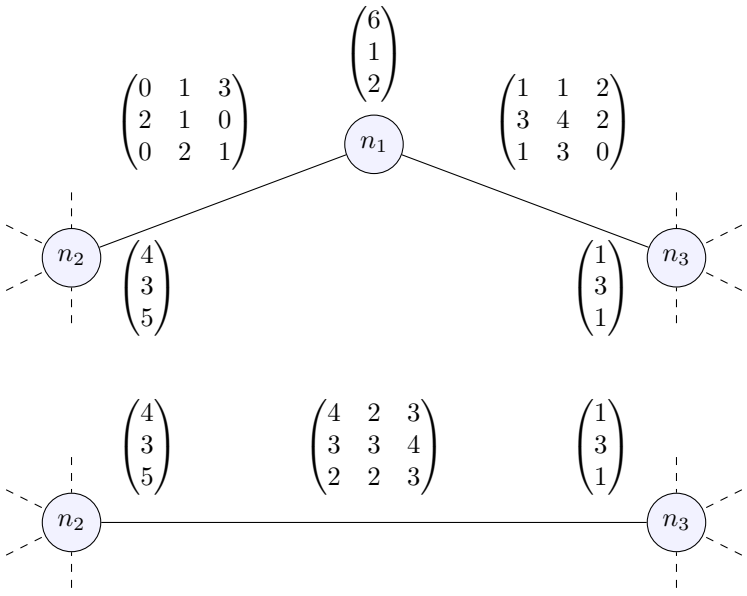


Abbildung 2.8: RII-Reduktion des Knoten n_1

Die beiden bisher vorgestellten RI- und RII-Reduktionen sind informationserhaltend, d.h. sie verschlechtern die Lösung gegenüber der optimalen Lösung nicht. Nun

kommen wir zu einer Reduktion, bei der die Lösung durch das Anwenden einer Heuristik auch schlechter werden kann. Idealerweise versuchen wir beim Lösen des PBQP diese Reduktion zu vermeiden und eine der beiden informationserhaltenden Reduktionen anzuwenden.

RN-Reduktionen – Entfernen von Knoten mit Grad größer Zwei

Die RN-Reduktion wird auf Knoten vom Grad größer Zwei angewendet, wenn zuvor alle möglichen RI- und RII-Reduktionen durchgeführt wurden, es also keine Knoten vom Grad Eins oder Zwei mehr gibt. Für die Auswahl einer Alternative wird hierbei eine Heuristik verwendet, die dazu führen kann, dass sich die tatsächliche Lösung der PBQP-Instanz gegenüber der optimalen Lösung verschlechtert. Allgemein kann frei entschieden werden welcher Knoten als nächstes reduziert wird. In unserem Fall entnimmt die Heuristik allerdings den nächsten zu reduzierenden Knoten aus einer Warteschlange und wählt für diesen unter Berücksichtigung der adjazenten Knoten die Alternative mit den geringsten Kosten. Danach werden die Kosten aller anderen Alternativen auf unendlich gesetzt und somit im Prinzip entfernt. Im nächsten Schritt wird dann versucht, den Knoten zu normalisieren. Da der Knoten nur noch eine Alternative besitzt, können alle inzidenten Kanten normalisiert werden.

Formal wird die Alternative $A_v \in \mathcal{A}_v$ gewählt, für die die Kostenfunktion

$$\sum_{u \in adj(v)} \min_{A_u \in \mathcal{A}_u} (c(A_u) + c(A_u, A_v)) + c(A_v) \tag{2.5}$$

minimal wird.

In Abbildung 2.9 sind keine Knoten mit Grad kleiner Drei vorhanden, also muss die Heuristik angewendet werden. Der nächste zu reduzierende Knoten sei n_1. Die Heuristik wählt als Lösung den dritten Eintrag des Kostenvektors, da dieser die geringsten Kosten verursacht.

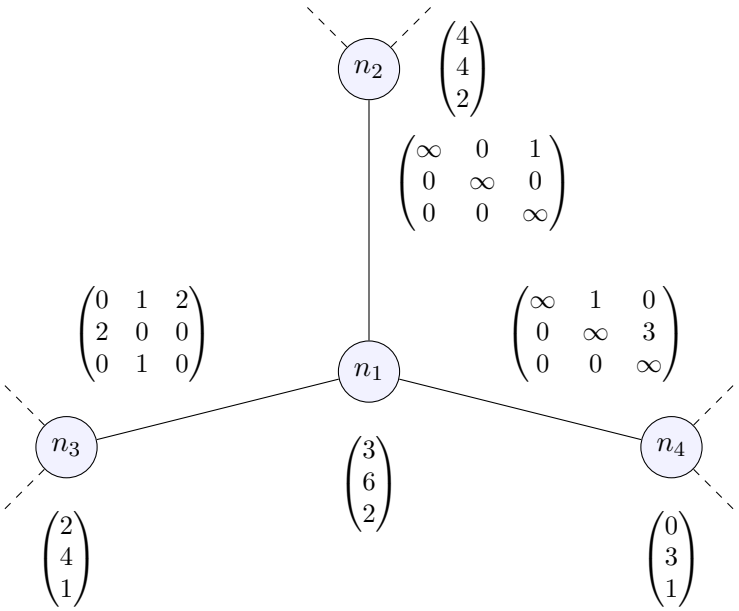


Abbildung 2.9: Knoten mit mehr als zwei inzidenten Kanten werden mit einer linearen Heuristik entfernt

3 Verwandte Arbeiten

In diesem Abschnitt werden einige mit dieser Arbeit verwandte Ansätze zur Registerzuteilung und Kopienminimierung vorgestellt.

Einer dieser Ansätze wird von Hames und Scholz in [HS06] behandelt. Wie auch in dieser Studienarbeit wird PBQP zur Lösung des Optimierungsproblems Registerzuteilung verwendet. Das Verfahren setzt allerdings keine SSA-Form als Zwischensprachendarstellung voraus. Dadurch ist die Anzahl der benötigten Register nicht vor der Registerzuteilung bestimmbar (siehe Abschnitt 2.4). Hames und Scholz beziehen daher die Möglichkeit zur Auslagerung von Registerwerten in den Lösungsvorgang mit ein. Kernpunkte der Arbeit sind die Einführung einer neuen Heuristik für RN-Reduktionen, sowie die Erweiterung des PBQP-Lösers um einen Branch-and-Bound-Algorithmus. Die neue Heuristik ist in der Lage, dynamisch eine Reduktionsreihenfolge für Knoten mit Grad größer Zwei zu bestimmen. Die Berechnung dieser Reihenfolge beruht auf der Färbbarkeit eines Knotens. Zunächst werden die Knoten absteigend nach ihrem Grad sortiert. Anschließend wird nach einem färbbaren Knoten gesucht. Ein Knoten ist dann färbbar, wenn der Grad des Knoten n kleiner als die Anzahl der verfügbaren Register k ist. Wird kein solcher Knoten gefunden, wird der Knoten mit dem geringsten Verhältnis zwischen Auslagerungskosten und Knotengrad reduziert. Die Ausführung von RN-Reduktionen in der so berechneten Reihenfolge führt laut Hames zu einer besseren Lösung. RI- und RII-Reduktionen werden aber immer, sofern möglich, zuerst durchgeführt. Der zweite Aspekt der Arbeit, die Erweiterung des PBQP-Lösers führt die Idee fort, dass eine optimale Lösung berechnet werden kann, indem bei RN-Reduktionen nicht eine einzelne Lösung ausgewählt wird, sondern alle möglichen Lösungen aufgezählt werden (siehe auch [HKS03]). Die Komplexität steigt dabei aber exponentiell mit der Anzahl der RN-reduzierten Knoten an. Hames greift diese Idee auf, indem der PBQP-Löser durch Branch-and-Bound-Techniken erweitert wird. Der Vorteil des Branch-and-Bound-Ansatzes ist, dass viele Lösungsmöglichkeiten wegfallen. Das Problem wird dazu in Teilprobleme zerlegt, die einen Suchbaum aufspannen. Ein Teilproblem in diesem Zusammenhang ist ein PBQP, das weder trivial noch durch RI- oder RII-Reduktionen gelöst werden kann.

Ein weiteres Verfahren, welches ebenfalls die Kopienminimierung auf SSA-basierten Zwischensprachen nicht als separate Phase behandelt, wird in [BMH10] vorgestellt. Das Verfahren versucht Quelle und Ziel von Kopierinstruktionen dasselbe Register zuzuweisen. Dazu wir der herkömmliche Algorithmus zur SSA-basierten Registerzuteilung durch einige Techniken erweitert. So wird für jede Variable eine Menge von bevorzugten Registern berechnet, die die Registerbeschränkungen der Variablen widerspiegeln. Weiter werden die gewählten Farben für das Ziel einer Kopierinstruk-

tion zur Quelle weiter propagiert. Als Letztes berechnet das Verfahren noch eine Reihenfolge der Grundblöcke deren Ziel es ist vorwiegend Kopierinstruktionen auf oft ausgeführten Programmpfaden zu entfernen. Ein weiterer Vorteil des Verfahrens besteht außerdem darin, dass der Interferenzgraph nicht länger benötigt wird und damit auch nicht aufgebaut werden muss.

Zwei weitere Ansätze zur Kopienminimierung werden in [Gru05] verfolgt. Einer dieser Ansätze verwendet eine Tauschheuristik, um eine gegebene Registerzuteilung zu verbessern und möglichst viele Kopien zu entfernen. Ähnlich wie in dieser Studienarbeit und in [HS06] transformiert der zweite Ansatz das Problem in ein mathematisches Optimierungsproblem, genauer in ein MILP (mixed integer linear programming).

Abschließend sei noch erwähnt, dass PBQP auch schon zur Lösung anderer Optimierungsprobleme in der Compilerentwicklung verwendet wurde. Beispielsweise wird PBQP zur Befehlsauswahl verwendet [EKS03].

4 Implementierung

4.1 Voraussetzungen

Das hier vorgestellte Verfahren setzt verschiedene Ergebnisse aus diversen Standardanalysen voraus. Dazu gehört der Aufbau des Dominanzbaums und eine Lebendigkeitsanalyse die für jede Variable die Lebendigkeitsinformationen bereit stellt. Um später die Entfernung von Kopien auf häufig ausgeführten Programmpfaden bevorzugen zu können, benötigen wir außerdem auch Informationen über die Ausführungshäufigkeiten der einzelnen Grundblöcke. Vorwiegend ist das Verfahren für IA32 als Zielarchitektur gedacht. Die Zuteilung der Register erfolgt dabei pro Registerklasse (für IA32: xmm, gp, vfp).

4.2 Abbildung auf PBQP

4.2.1 PBQP zur Registerzuteilung

Wie in Abschnitt 2.4 schon erwähnt, wird Registerzuteilung oft auf die Färbung eine Interferenzgraphen abgebildet. Registerzuteilung mittels PBQP lässt sich ebenfalls auf einen Graphen, den PBQP-Graphen abbilden. Dieser kann als Erweiterung eines Interferenzgraphen aufgefasst werden. Jedem Knoten wird dazu ein Kostenvektor und jeder Kante eine Kostenmatrix zugeordnet.

Definition 1 (Kostenvektor) *Der Kostenvektor $\vec{c}$ eines Knoten n ist gegeben durch:*

$$\vec{c_i} = \begin{cases} \infty & \text{, falls } r_i \text{ nicht an } n \text{ zuteilbar ist} \\ 0 & \text{, sonst} \end{cases} \tag{4.1}$$

wobei $\vec{c} \in \{0 \cup \infty\}^d$, $i \in 0, \ldots, d-1$.

Die Dimension d des Vektors ist gegeben durch die Anzahl der zuteilbaren Register der jeweiligen Zielarchitektur und der daraus betrachteten Registerklasse. Für jedes Register r_i gibt es also eine eindeutige Position i im Kostenvektor. Ist ein Register für einen bestimmten Knoten nicht zuteilbar so wird der Eintrag für das Register im Kostenvektor auf ∞ gesetzt. Ist es zuteilbar so erhält es die Kosten 0.

(a) (b)

Abbildung 4.1: (a) zeigt den strukturellen Aufbau eines PBQP-Knoten. (b) gibt ein konkretes Beispiel eines solchen Knoten an, bei dem Register r_0 verboten ist.

Definition 2 (Interferenzmatrix) *Die Kostenmatrix C für eine Interferenzkante zwischen zwei Knoten u und v wird definiert durch:*

$$C_{ij} = \begin{cases} \infty & , \textit{falls } i = j \\ 0 & , \textit{sonst} \end{cases} \tag{4.2}$$

wobei $C \in \{0 \cup \infty\}^{d \times d}$ und $i, j \in 0, \ldots, d-1$.

Diese Matrix erzwingt die Auswahl zweier unterschiedlicher Register r_i und r_j. Die Auswahl desselben Registers r_i für beide Knoten u und v würde unendliche Kosten verursachen.

$$\begin{pmatrix} \infty & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \infty \end{pmatrix}$$

Abbildung 4.2: Matrix für Interferenzkanten

Im Gegensatz zu anderen Ansätzen mit PBQP ([HS06]) die nicht auf SSA basieren, muss hier kein Auslagern von Registern in den Zuteilungsvorgang einbezogen werden.

Beispiel 2 *Abbildung 4.3 zeigt einen einfachen PBQP-Graphen mit drei Knoten die untereinander interferieren und von denen zwei Knoten mit Registerbeschränkungen versehen sind.*

Angenommen es stehen drei Register r_0, r_1 und r_2 zur Verfügung. Da die Knoten untereinander interferieren, werden auch mindestens drei Register benötigt (siehe Abschnitt 2.3). Knoten n_1 und n_3 sind in der Wahl der Register eingeschränkt. Knoten n_1 kann nur eines der ersten beiden Register zugeteilt bekommen, Knoten n_3 nur eines der letzten beiden. Knoten n_2 kann jedes der drei Register erhalten. Eine der möglichen Lösungen wäre beispielsweise: $n_1 = r_0; n_2 = r_2; n_3 = r_1$.

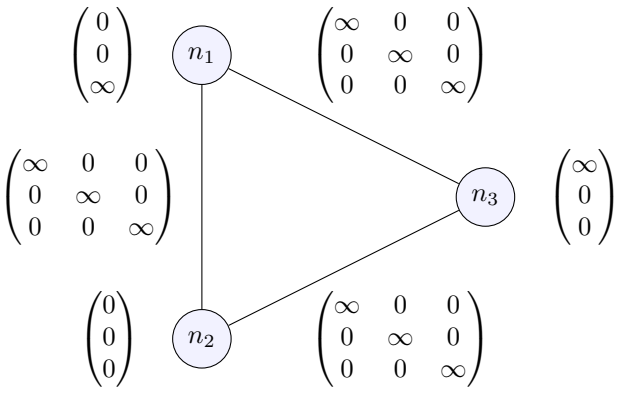


Abbildung 4.3: PBQP-Graph mit Interferenzkanten

4.2.2 PBQP zur Kopienminimierung

Der bisherige Aufbau berücksichtigt nur Interferenzen zwischen symbolischen Registern und eventuelle Beschränkungen der zuweisbaren Register. Nun soll der bisherige Ansatz erweitert werden, um Kopien wie sie beim SSA-Abbau und durch Registerbeschränkungen entstehen können zu minimieren. Dazu führen wir Affinitäten zwischen symbolischen Registern ein. Affinitäten schließen im Unterschied zu Interferenzen die Zuweisung desselben Registers nicht aus, sondern fördern diese. Kann kein gemeinsames Register zugeteilt werden, muss eine Kopierinstruktion eingefügt werden, die den Inhalt des ersten Registers im zweiten Register bereitstellt. Aber gerade diese Kopien sollen nach Möglichkeit vermieden werden.

Im PBQP-Graphen können solche Affinitäten durch Affinitätskanten ausgedrückt werden, die wie die Interferenzkanten eine spezielle Matrixform aufweisen.

Definition 3 (Affinitätsmatrix) *Die Kostenmatrix C für eine Affinitätskante zwischen zwei Knoten u und v wird definiert durch:*

$$C_{ij} = \begin{cases} 0 & , \textit{falls } i = j \\ const & , \textit{sonst} \end{cases} \tag{4.3}$$

wobei $C \in \{\mathbb{N} \cup \infty\}^{d \times d}$ für alle $i, j \in 0, \ldots, d-1$ und $const \in \mathbb{N}$.

Die Einträge auf der Diagonalen der Matrix werden auf 0 gesetzt, während alle anderen Einträge einen höheren ($const \in \mathbb{N}$), jedoch endlichen Wert erhalten. Dieser Wert kann entweder fest vorgegeben, oder auf Grundlage der voraussichtlichen oder bekannten Ausführungshäufigkeit für jede Matrix entsprechend angepasst werden. Innerhalb einer Matrix bleibt dieser allerdings konstant. Für die Lösung der PBQP-Instanz bedeutet dieser Aufbau, dass eine Auswahl desselben Registers kostengünstiger (Kosten 0) ist als die Auswahl unterschiedlicher Register. Die Auswahl eines anderen Registers wird aber auch nicht explizit verboten. Durch Anpassung der

Kosten nach Ausführungshäufigkeiten können Knoten, die aufgrund von Affinitäten dasselbe Register erhalten sollten gegenüber weniger oft ausgeführten Knoten bevorzugt werden.

$$C_{ij} = \begin{pmatrix} 0 & 2 & \dots & 2 \\ 2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 2 \\ 2 & \dots & 2 & 0 \end{pmatrix}$$

Abbildung 4.4: Matrix für Affinitätskanten

Beispiel 3 *Abbildung 4.5 zeigt einen Codeausschnitt und den zugehörigem PBQP-Graphen, wie er zur Registerzuteilung und Kopienminimierung aufgebaut werden würde. Anders als im vorherigen Beispiel sind hier neben Interferenzkanten auch Affinitätskanten vorhanden.*

Um am Ende des Codeausschnittes den richtigen Wert auszugeben, muss die Semantik der ϕ-Funktion nachgebildet werden. Dies kann entweder durch Registerkopien, die sicherstellen, dass sich der entsprechende Wert im Register von e_3 befindet, oder durch eine geschickte Registerwahl geschehen. Wird für e_1, e_2 und e_3 dasselbe Register verwendet, so ist keine Kopie erforderlich. Dies drücken die gestrichelten Affinitätskanten aus.

Bei drei zuteilbaren Registern r_0, r_1 und r_2 wäre eine sinnvolle Zuteilung beispielsweise: $a = r_0, b = r_1, c = r_0, d = r_1, e_1 = e_2 = e_3 = r_2$.

```
1  a = ...;
2  if(...) {
3    b = ...;
4    c = a + b;
5    d = b + c;
6    e1 = c + d;
7    print(d);
8  } else {
9    e2 = 5;
10   print(a);
11 }
12 e3 = φ(e1,e2);
13 print(e3);
```

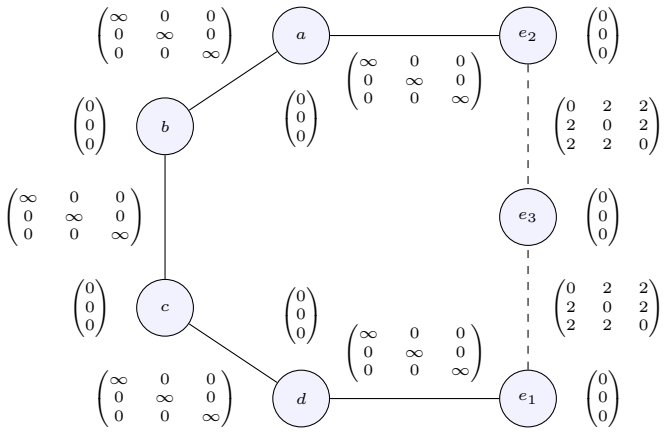


Abbildung 4.5: Codeausschnitt mit zugehörigem PBQP-Graph

4.3 Lösbarkeit

Wie wir in Abschnitt 2.3 und Abschnitt 2.4.2 schon gesehen haben, ist aufgrund der SSA-Form die maximale Anzahl der benötigten Register im Voraus bestimmbar und die Auslagerungsphase kann vor der Registerzuteilung den entsprechenden Auslagerungscode in das Programm einfügen. Danach ist sichergestellt, dass eine gültige Färbung mit der vorhanden Anzahl der Register existiert. Eine Färbung nennen wir in diesem Zusammenhang gültig, wenn die Summe der Kosten eines gelösten PBQP endlich ist.

Anschaulich bedeutet dies, dass an zwei interferierende Knoten nie dieselbe Farbe zugeteilt wurde. Auf SSA-Interferenzgraphen liefert eine Färbung in umgekehrter PES Reihenfolge eine solche Zuteilung. Für die Zuteilung während der Lösung des PBQP folgt daraus, dass man eine gültige Färbung erhält, wenn man die Knoten ebenfalls in einer solchen Reihenfolge färbt. Berechnet werden kann eine PES durch einen Post-Order-Durchlauf des Dominanzbaumes [HGG05].

Für Interferenzgraphen mit Knoten ohne Registerbeschränkungen funktioniert die Zuteilung in umgekehrter PES Reihenfolge, also immer. Sind Registerbeschränkungen vorhanden, kann dieses Vorgehen allerdings scheitern, da nicht alle Farben zur Verfügung stehen. Die beiden nachfolgenden Beispiele zeigen zwei solche Fälle.

Beispiel 4 *Gegeben sei der folgende Interferenzgraph mit fünf Knoten. Knoten e sei auf die ersten beiden Farben beschränkt, alle anderen Knoten seien unbeschränkt. Es stehen vier Farben (Rot, Gelb, Grün und Blau) zur Verfügung.*

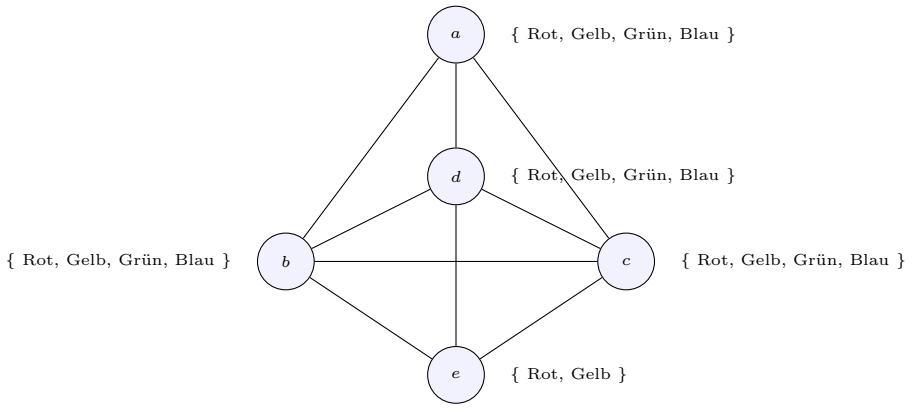


Abbildung 4.6: Färbung in umgekehrter PES scheitert bei schlechter Anfangswahl (z.B a = Grün)

Eine mögliche PES ist e, d, c, b, a. Da die größte Clique vier Knoten enthält, werden für eine Färbung mindestens vier Farben benötigt. Eine gültige Färbung kann also nur erreicht werden, wenn a und e dieselbe Farbe zugewiesen bekommen, da diese beiden Knoten als einzige nicht miteinander interferieren. Knoten a wird laut PES zuerst gefärbt (Färbung in umgekehrter Reihenfolge). Wählt man im ersten Schritt

für a eine Farbe (z.B Grün), die für e aber verboten ist, so ist keine gültige Färbung mehr möglich.

Beispiel 5 *Während eine gültige Färbung des Graphen im vorherigen Beispiel 4 mit dem richtigen PES möglich ist, kann für dieses Beispiel mit keinem PES eine Lösung gefunden werden.*

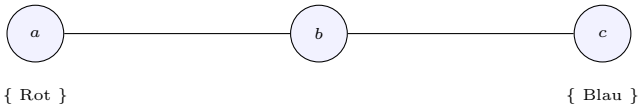


Abbildung 4.7: Die Beschränkungen von a und c verhindern eine Färbung mit der minimalen Anzahl an Farben

Die größte Clique des Graphen hat die Größe $\omega(G) = 2$. Demnach wären für eine minimale Färbung $\chi(G)$ des Graphen auch nur zwei Farben erforderlich. Die Beschränkungen der Knoten a und c verhindern dies aber, da für eine minimale Färbung a und c dieselbe Farbe erhalten müssten. Egal in welcher PES man färbt, man benötigt immer eine dritte Farbe für b.

In unserem Fall lässt sich aber dennoch immer eine gültige Färbung finden. Das Problem bei Registerbeschränkungen besteht darin, dass nicht für jeden Knoten alle Farben frei gewählt werden können. Wenn es aber nun möglich ist, die Register direkt vor einem solchen Knoten wieder frei verfügbar zu machen, so sind die für den Knoten benötigten Farben auch wieder vorhanden und der Knoten kann gefärbt werden. Eine Möglichkeit ist, die Werte der belegten Register in den Speicher auszulagern. Die andere Möglichkeit führt spezielle Permutations-Funktionen ein, die eine beliebige Permutation der Register ermöglichen [GH07]. Semantisch gesehen erhalten diese Permutations-Funktionen alle an dieser Stelle lebendigen Variablen und erzeugen daraus neue. Dadurch wird das Lebendigkeitsintervall jeder Variable beendet und ein neues entsteht. Dies hat Auswirkungen auf den Interferenzgraphen, welcher an diesen Stellen durchschnitten wird und sich dadurch mehrere Teilgraphen ergeben, für die folgendes gilt:

- Jeder Teilgraph ist mit der vorhanden Anzahl an Registern färbbar.
- Alle beschränkten Knoten sind Mitglieder derselben Clique.

Die Teilgraphen sind wie der komplette Interferenzgraph chordal, da keine neuen Kanten hinzugekommen sind und somit auch keine neuen Zyklen entstehen können. Also existiert auch ein perfektes Eliminationsschema. Da die beschränkten Knoten im Programmfluss direkt nach dem Schnitt stehen sind sie auch die letzten Knoten, die bei einem Post-Order-Durchlauf des Dominanzbaums in eine PES eingefügt werden. Damit wird die Clique mit den beschränkten Knoten aber auch als erstes

gefärbt. Da alle nachfolgende Knoten des Teilgraphen unbeschränkt sind, können diese ganz normal in umgekehrter PES Reihenfolge gefärbt werden.

Was wir nun brauchen ist also eine gültige Färbung der beschränkten Clique. Eine solche Färbung kann klassisch mittels bipartitem Matching ermittelt werden. Auf IA32 kann dieses Problem allerdings noch auf eine andere, hier verwendete Weise gelöst werden.

Registerbeschränkungen treten auf IA32-Architekturen nicht beliebig, sondern nur in bestimmten, nachfolgend genannten Konstellationen auf.

- nur ein mögliches Register: $(\infty, \infty, \infty, \infty, 0, \infty, \infty)$
- nur die ersten vier Register (edx,ecx,ebx,eax) sind möglich: $(0, 0, 0, 0, \infty, \infty, \infty)$
- alle Register sind möglich: $(0, 0, 0, 0, 0, 0, 0)$

Eine gültige Färbung kann jetzt ermittelt werden, wenn die Knoten in einer, nachfolgend beschriebenen Reihenfolge gefärbt werden:

- Teile die Knoten in zwei Gruppen ein: beschränkte und unbeschränkte
- Sortiere die Knoten innerhalb der Gruppe absteigend nach ihrem Interferenzgrad, also nach der Anzahl der mit dem Knoten interferierenden Nachbarn.

Eine solche Sortierung bevorzugt also alle beschränkten und stark interferierenden Knoten.

Nachfolgenden zeigen wir, dass aufgrund dieser speziellen Beschränkungen für IA32 immer ein gültige Färbung für Cliquen mit beschränkten Knoten gefunden wird, vorausgesetzt der Graph ist färbbar. Dies wird aber von der Auslagerungsphase sichergestellt.

Sei M die Menge der vorhandenen Farben. Dann entspricht die Menge aller möglichen Kombinationen aus zuteilbaren Farben der Potenzmenge $\mathcal{P}(M)$ von M und die Inklusionsrelation $\subseteq$ definiert ein Halbordung auf dieser Menge. Lässt man Beschränkungen auf eine Farbe außer acht, dann entspricht für IA32 die Halbordnung sogar einer Totalordnung, da es initial nur zwei Mengen gibt, die gesamte Menge M aller Farben und eine reduzierte Menge M', die nur die ersten vier Farben enthält. Für diese beiden Mengen gilt die Inklusionsrelation und sie definiert ein Totalordnung zwischen den Mengen, die auch erhalten beleibt, wenn man eine Farbe aus allen Mengen entfernt.

Wir wollen zeigen, dass auf jeden Fall eine Färbung gefunden wird, wenn wir immer den Knoten mit den meisten Beschränkungen zuerst färben. Dazu ordnen wir die Knoten jeweils der kleinsten Menge aus der obigen Totalordnung zu, die deren mögliche Farben enthalten.

Nun nehmen wir an, dass durch das Färben in der oben genannten Reihenfolge nicht immer eine gültige Färbung gefunden wird. Sei v der erste Knoten der nicht mehr

färbbar ist. Sei weiter n die Anzahl der möglichen Farben für v. Da v nicht mehr färbbar ist, müssen aufgrund der Inklusionsrelation zuvor schon n Knoten je eine der an v zuteilbaren Farben erhalten haben. v ist also der n+1 Knoten der eine der n Farben verwenden möchte. Damit ist die Clique nicht mehr färbbar, was ein Widerspruch zur Voraussetzung ist, dass die Clique färbbar ist.

Bleiben noch die zuvor ignorierten Knoten, denen nur eine bestimmte Farbe zugeteilt werden darf. Da für diese keine Wahlmöglichkeit besteht, müssen ihnen die geforderten Farben zugeteilt werden. Für die weitere Färbung der Clique stehen die Farben dann nicht mehr zur Verfügung und können daher aus allen oben genannten Mengen entfernt werden. Somit gilt weiterhin die Inklusionsrelation zwischen M und M' und die Totalordnung bleibt erhalten.

Das nachfolgende Beispiel 6 soll das Färben eines Graphen mit beschränkten Knoten nach obigem Schema noch einmal verdeutlichen:

Beispiel 6 *Betrachten wir noch einmal den Graphen aus Abbildung 4.6. Aus Gründen der Übersicht betrachten wir hier einen sehr kleinen Graphen und schränken deshalb auch die Anzahl der Register auf vier ein. Daher gelten für dieses Beispiel Knoten mit zwei frei wählbaren Registern als beschränkt.*

Wie wir gesehen haben ist der Graph aufgrund der Registerbeschränkung von Knoten e nicht mit jedem umgekehrten PES färbbar. Hier kommt nun unsere spezielle Sortierung der Knoten zum Tragen. Knoten e ist nach unserer Definition beschränkt. Daher müssen wir ihn zuerst färben. Er kommt ganz an den Anfang der Liste. Da nun keine weiteren beschränkten Knoten vorhanden sind müssen wir die restlichen Knoten nach ihrem Interferenz-Grad sortieren. Demnach sind b, c und d die nächsten zu färbenden Knoten, da sie einen höheren Grad als a haben. Wir fügen zunächst sie und dann a in die Liste ein. Jetzt haben wir unsere Reihenfolge zum Färben: e, b, c, d, a. Egal welche der beiden Möglichkeiten wir für e wählen, der restliche Graph bleibt immer färbbar.

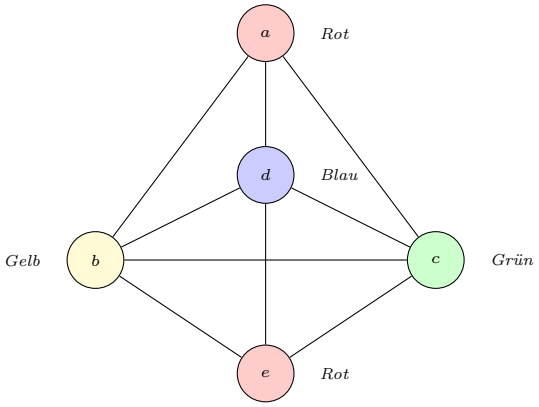


Abbildung 4.8: Erfolgreich gefärbter Graph – Färbung mit modifizierter Reihenfolge funktioniert trotz beschränktem Knoten

4.3.1 Bevorzugung von informationserhaltenden Transformationen

Wie wir gesehen haben, kann durch eine entsprechende Reihenfolge beim Färben das Finden einer gültigen Lösung garantiert werden. Leider kann dies aber auch dazu führen, dass RN-Reduktionen durchgeführt werden müssen, obwohl RI- oder RII-Reduktionen möglich wären. Was nun, wenn man in solchen Fällen die Reihenfolge ignoriert und die informationserhaltenden Transformationen durchführt sobald sie anwendbar sind? Um diese Frage zu beantworten, zeigen wir, dass die Reihenfolge der Reduktionen durch das vorzeitige Anwenden von RI- oder RII-Reduktionen gültig bleibt. Dazu behandeln wir die beiden Fälle RI- und RII-Reduktionen getrennt.

Für die Frage ob die PES nach einer RI-Reduktion außerhalb der Reihenfolge weiterhin gültig bleibt, betrachten wir einen Knoten u der durch eine RI-Reduktion reduziert werden soll und einen weiteren zu u adjazenten Knoten v. Zu unterscheiden sind zwei Fälle:

1. u **wird laut PES vor** v **reduziert:**
   Wird u laut PES vor v entfernt, so sind alle anderen zu u adjazenten Knoten schon zuvor entfernt worden, denn sonst hätte u nicht Grad Eins. Aus Sicht von u ist v also der letzte verbliebene Nachbar, analog zur Reduktion in PES Reihenfolge. Da v laut PES sowieso erst nach u entfernt wird, ändert sich auch hier die Situation nicht.

2. u **wird laut PES nach** v **reduziert:**
   Würde v ordnungsgemäß vor u reduziert, dann müssen zuerst alle anderen zu v adjazenten Knoten reduziert werden, da diese mit u keine Clique bilden. Danach kann v problemlos entfernt werden, da u keine weiteren Nachbarn besitzt und somit mit sich selbst eine Clique bildet. Wird u nun aber durch eine RI-Reduktion vor v reduziert, dann kann v trotzdem nach den anderen zu ihm adjazenten Knoten entfernt werden, wie es die PES ursprünglich vorsieht.

Für beide Fälle bleibt die PES also gültig und wir können RI-Reduktionen vorziehen.

Auch bei einer vorgezogenen RII-Reduktion bleibt die PES gültig. Dazu sei wieder u der zu reduzierende Knoten und v und w zwei zu u adjazente Knoten. Es gibt wieder zwei Fälle zu unterscheiden:

1. u **wird laut PES vor** v **und** w **reduziert:**
   In diesem Fall muss es zwischen v und w bereits ein Interferenzkante geben denn sonst könnte u nicht entfernt werden da sich v und w nicht in derselben Clique befinden. u kann also mittels RII-Reduktion reduziert und damit aus der PES entfernt werden.

2. u **wird laut PES nach** v **und entweder vor oder nach** w **reduziert:**
   Zunächst sei angenommen das zwischen v und w eine Interferenzkante existiert. Normalerweise würde v zuerst reduziert werden. Dazu müssen alle zu v adjazenten Knoten außer u und w zuerst reduziert werden, da sie nicht in derselben Clique wie u und w sind. Sonst hätte u nicht den Grad Zwei. v kann

dann reduziert werden, da u und w selbst auch Nachbarn sind. Wird nun u vorzeitig durch ein RII-Reduktion entfernt, kann v immer noch vor w reduziert werden, da w als einziger verbliebener Nachbar mit sich selbst eine Clique bildet. Die PES bleibt also gültig. Nun betrachten wird den Fall, dass zwischen v und w keine Kante existiert. Durch die vorzeitige RII-Reduktion von u wird zwischen v und w eine neue Kante erzeugt. Für v hat sich die Situation dennoch nicht verändert. Die Anzahl an maximalen Cliquen, in denen v Mitglied ist, ist gleich geblieben, ebenso wie der Knotengrad. Für w gilt dies ebenso. Da v laut PES vor w reduziert wird, also auch alle Nachbarn von v außer w schon reduziert sind, verhält sich die Reduktion von v aus Sicht von w, wie im oben beschriebenen Fall einer RI-Reduktion.

Die Fälle in den w vor v in der PES steht sind analog. Es ist also möglich sowohl RI- wie auch RII-Reduktionen ungeachtet der PES vorzuziehen und anzuwenden sobald sie möglich sind.

4.4 Aufbau und Lösungsverfahren

Der folgende Abschnitt beschreibt, wie ein PBQP auf Grundlage der beiden vorhergehenden Abschnitte konkret aufgebaut und gelöst wird. Dazu werden wir auch zwei unterschiedliche Varianten des Lösungsverfahren vorstellen.

4.4.1 Aufbau des PBQP

Der Aufbau der PBQP-Instanz ist für beide Lösungsvarianten gleich. Wir benötigen die zu färbenden Knoten, deren Beschränkungen, die Interferenz- und Affinitätskanten zwischen den Knoten, sowie eine Eliminationsreihenfolge in Form eines modifizierten umgekehrten perfekten Eliminationsschema.

Im ersten Schritt müssen die Knoten des Graphen erzeugt werden. Dazu wird für jedes symbolische Register ein neuer Knoten angelegt und der Kostenvektor, also die möglichen Registerbeschränkungen ermittelt und eingetragen.

Das Eliminationsschema und die Interferenzkanten können in einem Schritt erzeugt werden. Dazu wird der Dominanzbaum in Post-Order und die Grundblöcke von Ende zu Start durchlaufen. In jedem Schritt muss für das entsprechende symbolische Register geprüft werden, mit welchen anderen es interferiert und gegebenenfalls eine Interfernzkante zwischen den zugehörigen PBQP-Knoten eingefügt werden. Außerdem fügt man den aktuell betrachteten PBQP-Knoten als neuen Eintrag zum Eliminationsschema hinzu. Als Datenstruktur für das Eliminationsschema eignet sich eine einfache Liste aus PBQP-Knoten. Fügt man die Einträge immer an den Anfang der Liste erhält man gleich die später benötigte umgekehrte Reihenfolge. Eine Besonderheit bilden die im vorherigen Abschnitt eingeführten Perm-Knoten. Sie selbst erhalten kein Register, aber die von ihnen erzeugten neuen Werte. Hier kommt die spezielle Sortierung der Knoten zum Tragen. Für den Perm-Knoten müssen alle adja-

zenten nachfolgenden Knoten ermittelt werden. Danach sortiert man zunächst nach beschränkten und unbeschränkten Knoten und anschließend absteigend nach dem Grad ihrer Interferenz. Diese so sortierte Liste kann jetzt einfach zur Eliminationsliste hinzugefügt werden, indem sie an den Anfang der Liste angefügt wird.

Nachdem der Graph soweit aufgebaut ist, fehlen nur noch die Affinitätskanten. Dazu durchlaufen wir noch einmal die Knoten des Graphen in beliebiger Reihenfolge und prüfen, ob sie mögliche Kandidaten für Affinitätskanten sind. Mögliche Kandidaten sind die Knoten, bei denen Kopien entstehen können. Hier unterscheidet das Verfahren drei Fälle: ϕ-Knoten, Perm-Knoten und 2-Adress-Code.

Um bei ϕ-Knoten Registerkopien zu vermeiden, müssen die Argumente möglichst im selben Register gehalten werden, welches auch anschließend das Ergebnis der ϕ-Funktion aufnimmt. Daher benötigen wir Affinitätskanten zwischen dem PBQP-Knoten, der die ϕ-Funktion repräsentiert, und den Knoten für die Argumente.

Eine weitere potenzielle Quelle für Kopien sind die im vorhergehenden Abschnitt eingeführten Perm-Knoten. Diese wurden einfügt, um für vorhandene Werte eine Neuzuteilung zu ermöglichen. Erhalten die Werte vor und nach dem Perm-Knoten zwei unterschiedliche Register, so muss der Wert vom ersten in das zweite Register kopiert werden. Schafft man es aber, ausgehend von der Färbung des zweiten Knotens den ersten genauso zu färben, dann wird keine Kopie benötigt. Daher fügen wir auch zwischen den Argumenten und den jeweiligen Ergebnissen der Perm-Knoten Affinitätskanten ein.

Der letzte Fall behandelt 2-Adress-Code. Bei zwei Adress-Code werden nur zwei Register verwendet. Für Befehle mit einem Ziel und zwei Quellregistern wie beispielsweise die Addition, bedeutet das, dass das Zielregister und eines der Quellregister identisch sind. Um dies auszudrücken, fügen wir zwischen dem Ergebnis-Knoten der Operation und einem der Operanden-Knoten eine Affinitätskante ein.

Damit haben wir den PBQP-Graphen vollständig aufgebaut und können mit der Lösung des PBQP fortfahren.

4.4.2 Lösen des PBQP

Die beiden hier vorgestellten Lösungsvarianten orientieren sich an dem in Abschnitt 2.6.2 beschriebenen, grundlegenden Lösungsalgorithmus für PBQP. Lediglich die Handhabung von Knoten mit Grad größer Zwei, also solche die durch ein RN-Reduktion reduziert werden, unterscheidet sich. Eine ausführliche Beschreibung des PBQP-Lösers findet sich in [Eck03, BZ08]. Hier soll der Lösungsvorgang nur noch einmal informell beschrieben werden. Das Hauptaugenmerk liegt auf den Ergänzungen des PBQP-Lösers, die für dieses Verfahren notwendig sind.

Zunächst führt der Algorithmus eine Normalisierung aller Kanten durch. Danach werden, soweit möglich, RI- und RII-Reduktionen angewendet, da diese den Graphen informationserhaltend transformieren. Reduziert werden die Knoten, indem die

inzidenten Kanten bei den Nachbarknoten abgetrennt werden. Der betrachtete Knoten behält die Information über seine Nachbarn. Diese wird später noch benötigt. Die so reduzierten Knoten werden dann auf dem Reduktionsstack zur späteren Färbung abgelegt. Sind keine informationserhaltenden Reduktionen mehr möglich, müssen solange RN-Reduktionen durchgeführt werden, bis wieder RI- oder RII-Reduktionen anwendbar sind oder alle Knoten reduziert wurden. Dazu unterschieden wir die folgenden beiden Varianten.

Frühes Färben Diese Variante wählt bei einer RN-Reduktion sofort eine minimale Lösung (also ein Register bzw. eine Farbe) aus und entfernt dann den Knoten aus dem Graphen, wodurch sich der Grad der Nachbarknoten verringert. Hierdurch können eventuell schon wieder RI- oder RII-Reduktionen möglich werden. Ist dies nicht der Fall, müssen weitere RN-Reduktionen durchgeführt werden. Das Auswählen einer minimalen Lösung erfolgt, wie in Abschnitt 2.6.2 beschrieben, durch Vergleichen der aufsummierten Kosten, die die Wahl einer der möglichen Alternative im Ausgangsknoten und den Nachbarknoten verursacht (vgl. Gleichung 2.5). Beachtet werden muss bei dieser Variante, dass die reduzierten Knoten gefärbt werden, weswegen die Reihenfolge der Reduktionen der Färbereihenfolge entsprechen muss.

Spätes Färben Bei der zweiten Variante wird, im Gegensatz zur Ersten bei Anwendung einer RN-Reduktion noch keine Farbe bzw. kein Register für den Knoten ausgewählt. Dieser wird lediglich durch das Entfernen der Kanten an den Nachbarknoten reduziert und wie die durch eine RI- oder RII-Reduktion entfernten Knoten auf dem Reduktionsstack abgelegt. Das Färben geschieht dann ebenfalls in der anschließenden Rückpropagationsphase. Somit haben die Nachbarknoten für den Moment noch eine Farbe mehr zur Auswahl. Da die Knoten nicht gefärbt sondern lediglich reduziert werden, müssen die Reduktionen in umgekehrter Färbereihenfolge durchgeführt werden.

Die Rückpropagationsphase beginnt, nachdem der Graph vollständig reduziert wurde. Es werden nacheinander Knoten vom Reduktionsstack genommen und für diese eine minimale Lösung gewählt. Dazu addiert man für jeden noch bekannten Nachbarn diejenige Spalte bzw. Zeile der Kostenmatrix, die durch die gewählte Lösung im Nachbarknoten vorgegeben wird, zum eigenen Kostenvektor hinzu und wählt anschießend den Eintrag mit den geringsten Kosten. Dadurch, dass der Knoten nur seine Nachbarn zum Zeitpunkt der Reduktion kennt und die Umkehr der Färbereihenfolge durch den Stack, wird sichergestellt, dass für alle bekannten Nachbarn schon eine Lösung ausgewählt wurde. Zum besseren Verständnis ist der Algorithmus in Listing 4.1 noch einmal als Pseudocode angegeben.

```
1  while(IsNotEmpty(reductionStack)) {
2    Node node = Pop(reductionStack);
3    vector vec = node.Vector;
4    foreach neighbor in get_neighbors(node) {
5      matrix mat = get_matrix(node,neighbor);
6      if(node_is_source(node)) {
7        neighbor = add_matrix_row(mat, node.solution);
8      } else {
9        neighbor = add_matrix_col(mat, node.solution);
10     }
11   }
12   node.Solution = get_min_index(vec);
13 }
```

Listing 4.1: Rückpropagation-Algorithmus

Wichtig für die Lösbarkeit der PBQP-Instanz bzw. die Gültigkeit einer Lösung ist, dass die Knoten in einer, wie im vorherigen Abschnitt beschriebenen Reihenfolge (umgekehrtes perfektes Eliminationsschema) reduziert werden. Dazu wird die Auswahl des nächsten zu reduzierende Knoten um eine Liste erweitert, die diese Reihenfolge enthält. Beachtet werden muss dabei, dass die Liste die Knoten in der Reihenfolge enthält, in der sie gefärbt werden müssen. Da die erste Variante des Verfahrens bei RN-Reduktionen sofort färbt, kann die Liste vom Anfang zum Ende abgearbeitet werden. Die zweite Variante verhält sich gerade umgekehrt. Die Knoten werden nur reduziert, aber noch nicht gefärbt. Da der Reduktionsstack durch seine LIFO-Eigenschaft die Reduktionsreihenfolge aber invertiert, müssen die Knoten in entgegengesetzter Reihenfolge, also vom Ende zum Anfang reduziert werden. Ausgenommen hiervon sind, wie schon mehrfach erwähnt, Knoten vom Grad kleiner Drei. Diese werden so früh wie möglich reduziert und müssen dann aus der Liste entfernt werden.

Die Gesamtlaufzeit des PBQP-Lösers liegt in $O(nb^3)$, wobei n die Anzahl der Knoten und b die Größe des Kostenvektors, also die Anzahl der Register in der jeweiligen Registerklasse angibt. Für eine gegebene Architektur ist das Verfahren linear in der Anzahl der Knoten da die Registeranzahl b konstant ist. Durch den kubischen Aufwand der Registerzahl eignet sich dieses Verfahren eher für Architekturen mit wenigen Registern, wie beispielsweise IA32.

4.5 Vollständiges Beispiel

4.5.1 Einführung und Aufbau

Im Folgenden soll eine Registerzuteilung anhand des vorgestellten Verfahrens (hier die erste Variante mit frühem Färben) demonstriert werden. Dazu betrachten wir das Beispiel aus Abschnitt 4.2.2. Es wurde ein wenig erweitert, um auch RN-Reduktionen demonstrieren zu können.

```
1  a = ...;
2  if(...) {
3    b = ...;
4    c = a + b;
5    d = b + c;
6    print(b);
7    print(a)
8    e1 = c + d;
9    print(d);
10 } else {
11   e2 = 5;
12   print(a);
13 }
14 e3 = phi(e1, e2);
15 print(e3);
```

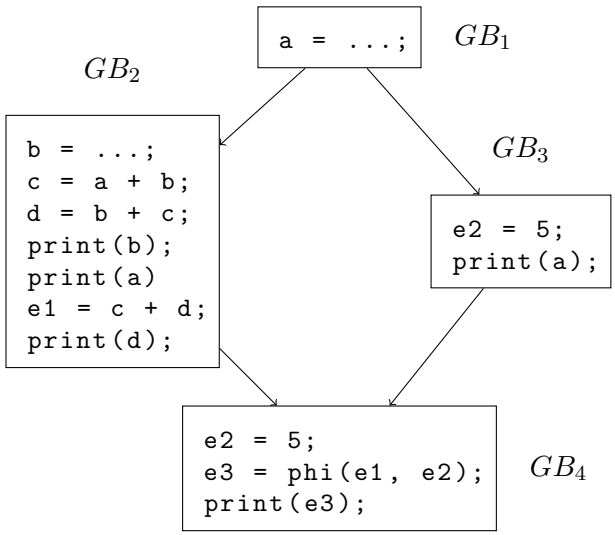


Abbildung 4.9: Beispielprogramm mit zugehörigem Steuerflussgraphen

Zunächst muss der PBQP-Graph sowie die Reduktionsreihenfolge aufgebaut werden. Wir durchlaufen dazu den Dominanzbaum in Post-Order-Reihenfolge, also z.B. GB_4, GB_3, GB_2, GB_1. Die Anweisungen in den Grundblöcken selbst werden ebenfalls in umgekehrter Reihenfolge durchlaufen (vom Ende zum Anfang). In jedem Schritt werden die lebendigen Variablen ermittelt. Kommen neue hinzu, so wird dem PBQP-Graphen ein entsprechender Knoten, sowie Interferenzkanten zu allen anderen momentan lebendigen Variablen hinzugefügt. Gleichzeitig wird das UPES (Umgekehrtes perfektes Eliminationsschema berechnet, indem bei jeder Definition der zugehörige PBQP-Knoten an den Anfang der PES angefügt wird. Eine Sortierung ist in diesem Beispiel nicht nötig, da keine Schnitte eingefügt werden müssen. Nun fehlen noch die Affinitätskanten. Dazu prüfen wir für jeden Knoten, ob er einem der drei Fälle für Adjazenzkanten entspricht. Falls ja, fügen wir die entsprechenden Adjazenzkanten ein, sofern diese noch nicht existieren. In unserem Beispiel ist der Knoten $e3$ als Repräsentant der ϕ-Funktion ein solcher Kandidat. Daher fügen wir zwischen ihm und den beiden Knoten $e1$ und $e2$, die den Argumenten der $\phi - Funktion$ entsprechen, je eine Affinitätskante ein. Nun haben wir den kompletten PBQP-Graphen aufgebaut (siehe Abbildung 4.10).

Es ergibt sich die folgenden Färbereihenfolge (UPES): $a, b, c, d, e1, e2, e3$.

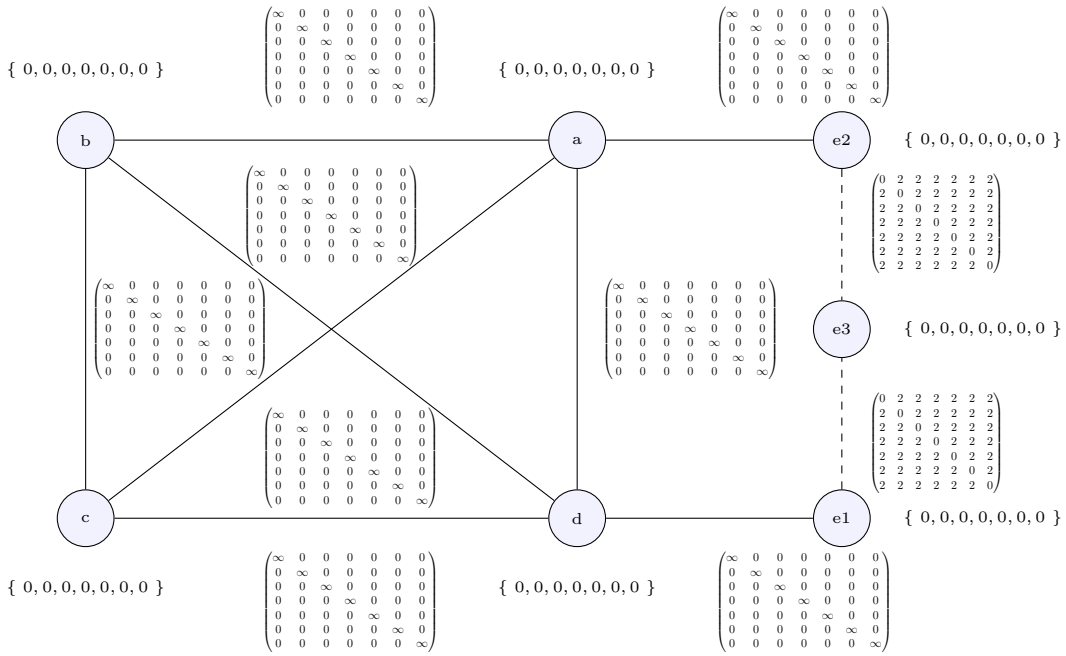


Abbildung 4.10: PBQP-Graph

4.5.2 Lösungsvorgang

Die erste Reduktion ist eine RII-Reduktion des Knoten $e3$ (siehe Abbildung 4.11). Dieser wird auf dem Reduktionsstack zur späteren Bearbeitung abgelegt.

Nach dieser Reduktion kann der nächste Knoten $e2$ ebenfalls mit einer RII-Reduktion entfernt (siehe Abbildung 4.12) und auf dem Reduktionsstack abgelegt werden.

Anschließend wird Knoten e1 reduziert (siehe Abbildung 4.13).

Wie man feststellt gibt es nun keine Knoten mit einem Grad kleiner Drei. Mit RI- oder RII-Reduktionen kommt man also nicht mehr weiter und muss nun eine RN-Reduktion durchführen. Das verbliebene UPES (a,b,c,d) besagt, dass als nächstes der Knoten a reduziert werden muss. Die RN-Reduktion wählt für a eine minimale Lösung und setzt alle anderen Einträge auf ∞. In unserem Fall wird der erste Eintrag des Kostenvektors ausgewählt. Da nun alle Vektoreinträge von a bis auf den Ausgewählten verboten sind, ist die Kante unabhängig und kann nach einer Normalisierung entfernt werden. Es bleibt der Graph in Abbildung 4.14.

Jetzt sind wieder RII-Reduktionen möglich und wir können den Knoten b reduzieren. Dieser landet wieder auf dem Reduktionsstack.

Die letzte Reduktion ist nun eine RI-Reduktion des Knoten d (siehe Abbildung 4.15). Wie auch bei den RII-Reduktionen wird der entfernte Knoten auf den Reduktionsstack gelegt. Übrig bleibt nur noch Knoten c, der zu keinem anderen Knoten adjazent

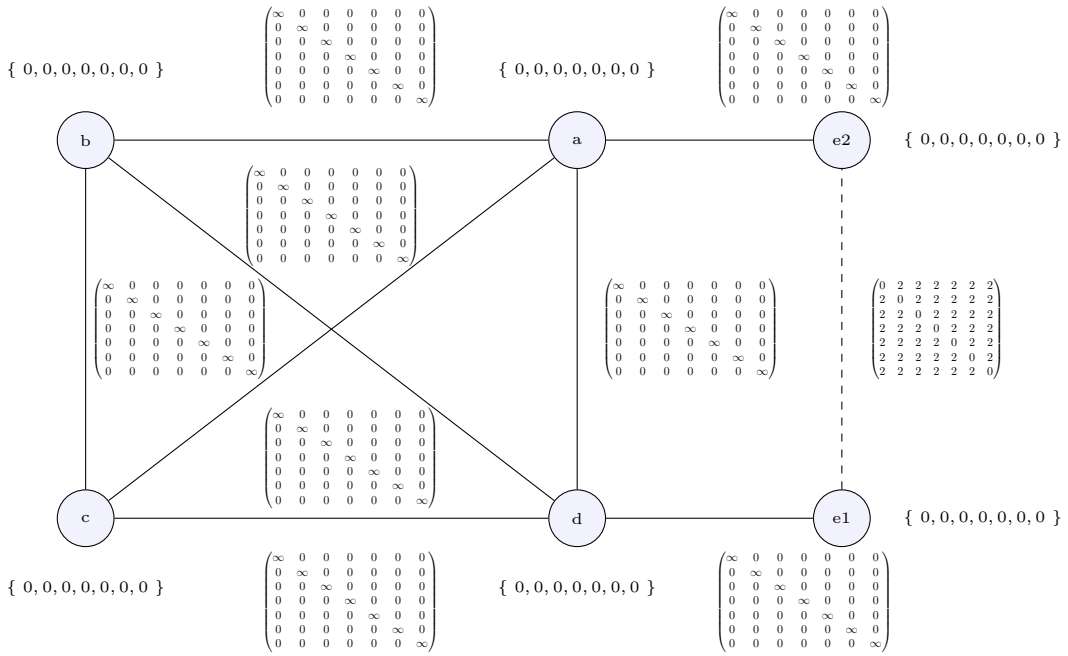


Abbildung 4.11: PBQP-Graph nachdem e3 reduziert wurde

ist. Für diesen kann eine triviale Lösung (R0-Reduktion) anhand seines eigenen Kostenvektors gefunden werden. Da nur der erste Eintrag verboten ist, können wir den Zweiten auswählen und dem Knoten damit das Register 1 zuweisen. Nun beginnt die Rückpropagationsphase, die für die Knoten auf dem Reduktionsstack eine Lösung ermittelt. Knoten d erhält Register 2, Knoten b das Register 3, Knoten $e1$, $e2$ und $e3$ das Register 1. Für Knoten a haben wir ja schon zuvor das Register 0 gewählt.

Somit haben wir eine gültige Lösung (keine interferierenden Knoten haben dasselbe Register) gefunden und auch Registerkopien vermieden, indem wir den Knoten $e1$, $e2$ und $e3$ dasselbe Register zugeteilt haben. In Abbildung 4.16 ist der gefärbte Graph noch einmal dargestellt.

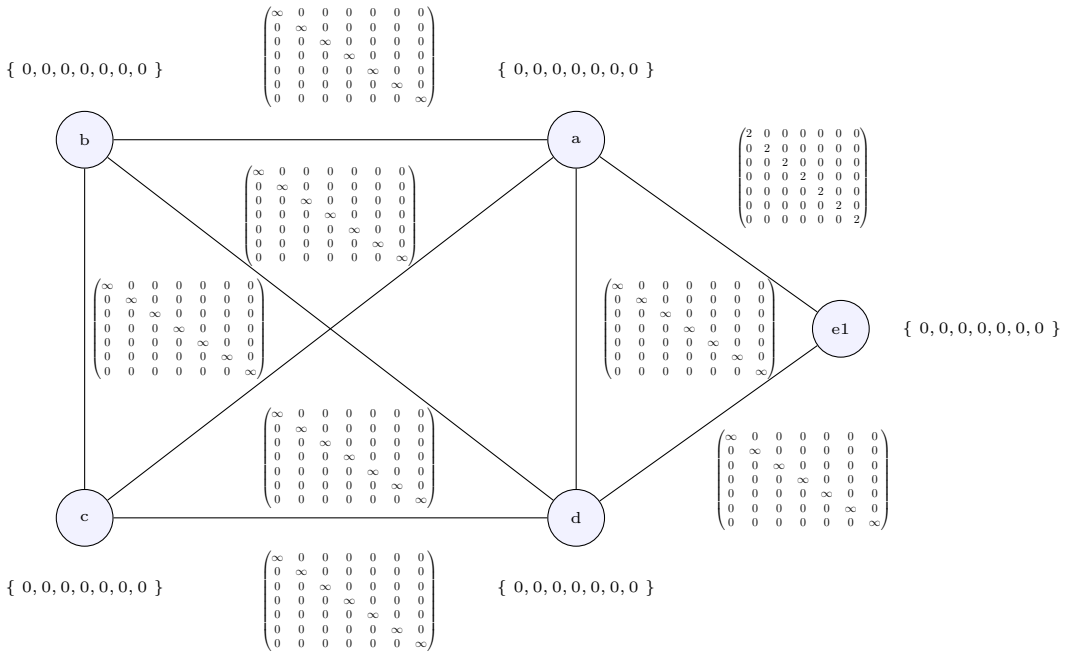


Abbildung 4.12: PBQP-Graph nachdem e2 reduziert wurde

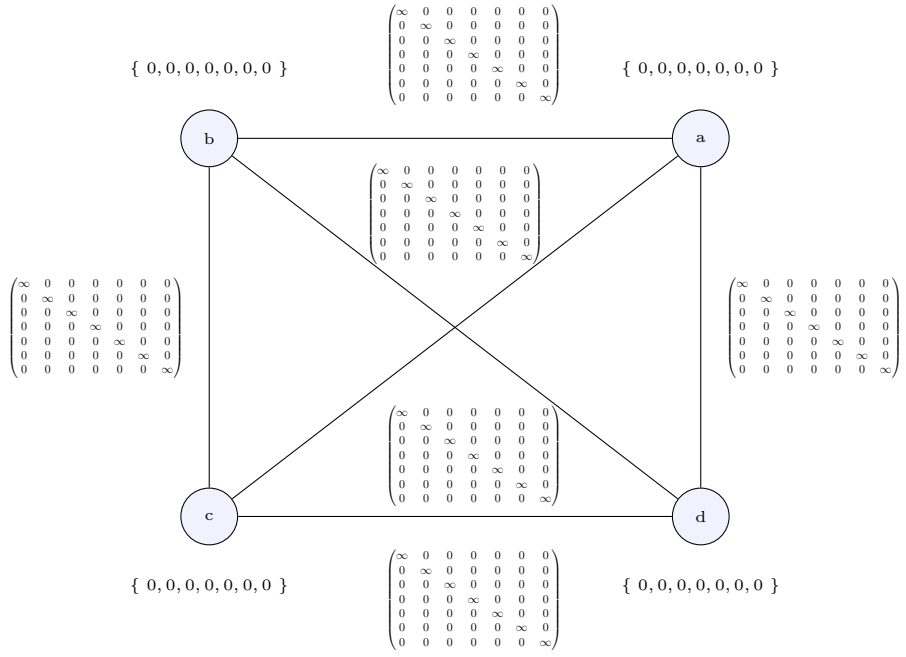


Abbildung 4.13: PBQP-Graph nachdem e1 reduziert wurde

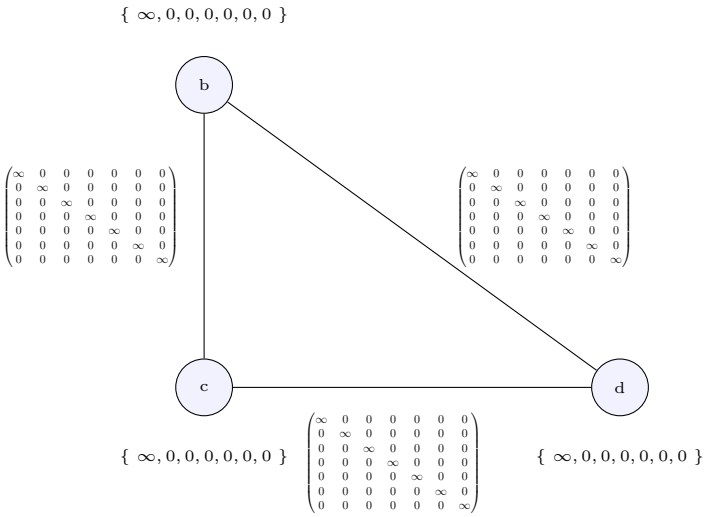


Abbildung 4.14: PBQP-Graph nachdem a reduziert wurde

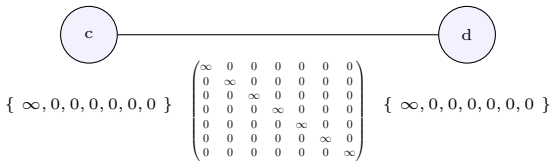


Abbildung 4.15: PBQP-Graph nachdem b reduziert wurde

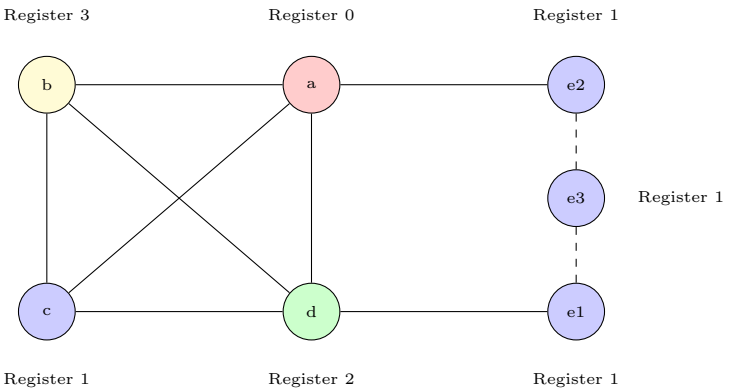


Abbildung 4.16: Gefärbter Graph bzw. zugeteilte Register

5 Bewertung

In diesem Abschnitt werden einige unter Firm vorhanden Verfahren zur Kopienminimierung mit den beiden in dieser Arbeit vorgestellten neuen Verfahren, hinsichtlich Lösungsqualität und benötigter Übersetzungszeit verglichen. Testgrundlage bilden die SPEC CINT2000 Benchmarks. Als Architektur kommt IA32 zum Einsatz. Das Testsystem bildet ein Intel Pentium Dual-Core E5300 mit einem Linux Betriebssystem (2.6.31-17-generic GNU/Linux) und einer festgelegten Taktfrequenz von 1200 MHz.

Heur4 ist das in Firm eingesetzte Standardverfahren zur Kopienminimierung [HG08]. Das Verfahren basiert darauf, eine gegebene Färbung des Interferenzgraphen zu verbessern, indem es Gruppen, sogenannten Chunks, von affinen Knoten bildet und anschließend versucht, den Knoten einer Gruppe möglichst dieselbe Farbe zuzuordnen.

5.1 Lösungsqualität

Tabelle 5.1 zeigt die Ausführungszeiten der einzelnen SPEC Testprogramme nach ihrer Erstellung. Die zweite Spalte gibt die Ausführungszeiten der Testprogramme nach einer Übersetzung ohne jegliche Kopienminimierung an. Die dritte Spalte zeigt die Ausführungszeiten von Heur4, die vierte Spalte die Ausführungszeiten von PBQP mit frühem Färben, sowie die Differenzen zur Heur4 und die letzte Spalte zeigt die Ausführungszeiten für PBQP mit spätem Färben sowie ebenfalls die Differenzen zur Heur4.

Die Tabelle zeigt, dass die Lösungsqualität der ersten PBQP-Variante mit frühem Färben durchaus mit der von Heur4 vergleichbar, oder in vielen Fällen sogar leicht besser ist. In einem Fall, bei 300.wolf ist es sogar um 4,2% besser. Die restlichen Testfälle unterscheiden sich nicht so stark und im Schnitt ergibt sich ein Unterschied von 0,4%. Beide Verfahren liefern also in etwa die gleiche Qualität. Die zweite PBQP-Variante mit spätem Färben hingegen, liefert etwas schlechtere Ergebnisse und ist im Schnitt 2,7% langsamer als Heur4. Da Hames in [HS06] mit dem Ansatz, Knoten erst spät zu färben gute Ergebnisse erzielen konnte, hätte man hier die in zwei Fällen (253.perlbmk und 300.wolf) deutlich schlechter Leistung des gleichen Ansatzes nicht erwartet. Die Messungen zeigen aber auch, dass es dennoch auch auf SSA möglich ist, mit PBQP eine qualitativ gute Registerzuteilung zu finden und eine hohe Zahl von Kopien zu vermeiden.

			PBQP			
Benchmark	Ohne CopyMin	Heur4	Frühes Färben		Spätes Färben	
164.gzip	137,0	103,2	103,9	+0,6%	104,8	+1,5%
175.vpr	95,3	93,2	93,5	+0,2%	93,6	+0,4%
176.gcc	54,0	49,4	49,4	-0,1%	50,8	+2,7%
181.mcf	99,1	98,0	98,7	+0,7%	98,4	+0,4%
186.crafty	61,7	53,0	52,3	-1,3%	53,4	+0,8%
197.parser	134,0	125,2	124,5	-0,5%	125,0	-0,1%
253.perlbmk	123,0	82,7	82,9	+0,2%	96,4	+14,2%
254.gap	66,0	58,9	58,9	-0,1%	58,8	-0,1%
255.vortex	119,0	110,5	110,8	-0,3%	112,4	+1,8%
256.bzip2	123,0	98,1	97,5	-0,6%	98,3	+0,2%
300.wolf	155,0	115,9	111,2	-4,2%	125,2	+7,4%
Durchschnitt:				-0,4%		+2,7%

Tabelle 5.1: SPEC CINT2000 Benchmark-Ergebnisse in Sekunden

5.2 Übersetzungszeit

Auch bei der Übersetzungszeit kann sich PBQP mit Heur4 auf IA32 messen. Tabelle 5.2 zeigt die Übersetzungzeiten, die der Firm-Compiler zum Übersetzen der Benchmarkprogramme unter Verwendung des jeweiligen Verfahrens benötigt. Bei beiden PBQP-Varianten sind zusätzlich die Differenzen zur Heur4 angegeben. Im Mittel ist die Übersetzungszeit der ersten PBQP-Variante mit der von Heur4 vergleichbar. Etwas anders verhält sich die zweite PBQP-Variante. Diese konnte jedes Testprogramm schneller als Heur4 übersetzen und erreichte eine durchschnittliche Beschleunigung von 4,75%. Der Geschwindigkeitsvorteil gegenüber der PBQP-Variante mit frühem Färben ergibt sich daraus, dass die Auswahl einer Lösung in der Rückpropagationsphase schneller möglich ist als bei der RN-Reduktion selbst. In der Rückpropagationsphase muss der Algorithmus lediglich die entsprechende Spalte oder Zeile der Kostenmatrix zum Kostenvektor des Knotens addieren und anschließend das Minimum des Vektors bestimmen. Bei der Wahl innerhalb einer RN-Reduktion hingegen, muss für jede Kombination aus lokalen Alternativen und den wählbaren Alternativen der Nachbarknoten geprüft werden, ob sie das Minimum ist. Dies verursacht deutlich höheren Aufwand, wodurch sich auch die schlechteren Übersetzungszeiten erklären lassen.

In Tabelle 5.3 sind die Ausführungszeiten der Phasen Registerzuteilung und Kopienminimierung aufgelistet. Im Gegensatz zu Heur4 benötigt PBQP keine separate Phase zur Kopienminimierung. Erstaunlicherweise ist im Fall der ersten PBQP-Variante die Ausführungszeit der eigentlichen Phase immer langsamer als die Summe aus einfacher Registerzuteilung und Heur4, obwohl die Gesamtübersetzungszeiten oftmals etwas schneller, im Mittel aber beinahe gleich sind. Daher müssen spätere Phasen des Compilers durch die PBQP-Lösung beschleunigt worden sein. Die zweite PBQP-Variante ist aber auch hier immer schneller als die beiden anderen.

		PBQP			
Benchmark	Heur4	Frühes Färben		Spätes Färben	
164.gzip	7.057	7.011	- 0,65%	6.457	- 8,5%
175.vpr	23.806	23.803	- 0,01%	22.704	- 4,63%
176.gcc	412.364	400.484	- 2,90%	390.837	- 5,22%
181.mcf	1.389	1.597	+ 14,00%	1.323	- 4,75%
186.crafty	35.769	34.301	- 4,10%	33.314	- 6,86%
197.parser	42.046	40.708	- 3,18%	40.916	- 2,69%
253.perlbmk	169.058	164.589	- 2,64%	162.537	- 3,86%
254.gap	127.684	128.688	+ 0,79%	122.652	- 3,94%
255.vortex	130.156	125.927	- 3,25%	121.854	- 6,38%
256.bzip2	5.203	5.261	+ 1,11%	5.112	- 1,75%
300.wolf	34.863	35.057	+ 0,55%	33.597	- 3,63%
Durchschnitt:			-0,03%		-4,75%

Tabelle 5.2: SPEC CINT2000 Übersetzungszeiten in Millisekunden

		PBQP			
Benchmark	Heur4	Frühes Färben		Spätes Färben	
164.gzip	135	930	1.065	1.320	696
175.vpr	376	2.334	2.710	3.367	2.235
176.gcc	8.018	35.350	43.368	47.000	35.160
181.mcf	36	147	183	316	182
186.crafty	568	3.696	4.264	4.103	3.148
197.parser	942	3.802	4.744	5.337	4.221
253.perlbmk	3.120	14.045	17.165	18.941	14.922
254.gap	2.656	12.120	14.776	18.260	12.320
255.vortex	2.267	12.521	14.788	16.847	13.091
256.bzip2	87	447	534	687	466
300.wolf	744	3.163	3.907	4.417	3.050

Tabelle 5.3: Dauer der einzelnen Phasen in Millisekunden

6 Zusammenfassung

In dieser Arbeit haben wir ein Verfahren für SSA-basierte Zwischensprachen vorgestellt, das sowohl Registerzuteilung als auch Kopienminimierung in einer einzigen Phase ermöglicht. Grundlage des Verfahrens bildet das Partitioned Boolean Quadratic Problem (PBQP). Durch die Erweiterung des Konzepts der Interferenzgraphen und die Einführung zweier spezieller Kantentypen, konnten wir eine Abbildung angeben, mit der PBQP zur Registerzuteilung und Kopienminimierung eingesetzt werden kann. Außerdem haben wir gezeigt unter welchen Voraussetzungen eine gültige Lösung garantiert gefunden wird. Kernstück dessen ist eine spezielle Reihenfolge, in der die Zuteilung erfolgen muss.

Weiterhin haben wir auch die Idee von Hames aufgegriffen, die Färbung nicht sofort bei der heuristischen Reduktion, sondern genau wie bei den informationserhaltenden Transformationen erst in der Rückpropagationsphase festzulegen. Dazu haben wir das ursprüngliche Verfahren um diese Variante erweitert.

Durch die praktische Umsetzung und Einbettung beider Varianten des Verfahrens in libFirm konnten wir die Leistungsfähigkeit überprüfen und beide Varianten sowohl untereinander als und auch mit anderen Verfahren aus libFirm vergleichen. Beim Vergleich der beiden PBQP-Varianten untereinander ergab sich, dass die erste Variante mit frühem Färben eine bessere Lösungsqualität, aber schlechtere Übersetzungszeiten als die zweite Variante mit spätem Färben liefert. Im Vergleich mit dem Standardverfahren von libFirm ergab sich eine leicht bessere Lösungsqualität durch die erste Variante des PBQP-Verfahrens. Die zweite Variante blieb knapp dahinter. Die Übersetzungszeiten konnten durch PBQP zumindest für IA32 Architekturen in beiden Fällen verbessert werden. Auf anderen Architekturen wie beispielsweise ARM, die mehr Register als IA32 zur Verfügung haben, ist aber zu erwarten, dass der kubische Aufwand in der Registeranzahl deutlich mehr zum Tragen kommt.

Literaturverzeichnis

[BCT94] Briggs, Preston ; Cooper, Keith D. ; Torczon, Linda: Improvements to graph coloring register allocation. In: *ACM Trans. Program. Lang. Syst.* 16 (1994), Nr. 3, S. 428–455. – ISSN 0164–0925

[BDR07] Bouchez, Florent ; Darte, Alain ; Rastello, Fabrice: On the Complexity of Register Coalescing. In: *CGO '07: Proceedings of the International Symposium on Code Generation and Optimization*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2764–7, S. 102–114

[BMH10] Braun, Matthias ; Mallon, Christoph ; Hack, Sebastian: Preference-Guided Register Assignment. In: *Compiler Construction* Bd. 6011, Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–11969–9, S. 205–223

[BZ08] Buchwald, Sebastian ; Zwinkau, Andreas: *Befehlsauswahl auf expliziten Abhängigkeitsgraphen*, Universität Karlsruhe (TH), IPD Goos, Diplomarbeit, Dec 2008

[Cha04] Chaitin, Gregory: Register allocation and spilling via graph coloring. In: *SIGPLAN Not.* 39 (2004), Nr. 4, S. 66–74. – ISSN 0362–1340

[Eck03] Eckstein, Erik: *Code optimizations for digital signal processors*, TU Wien, Diss., November 2003

[EKS03] Eckstein, Erik ; König, Oliver ; Scholz, Bernhard: Code Instruction Selection Based on SSA-Graphs. 2003, S. 49–65

[GH07] Grund, Daniel ; Hack, Sebastian: A Fast Cutting-Plane Algorithm for Optimal Coalescing. In: *Compiler Construction 2007* Bd. 4420, Springer, March 2007, S. 111–125

[Gru05] Grund, Daniel: *Kopienminimierung in einem SSA-basierten Registerzuteiler*, Universität Karlsruhe, Diplomarbeit, August 2005

[Hac05] Hack, Sebastian: Interference Graphs of Programs in SSA-form. 2005. – Forschungsbericht

[HG08] Hack, Sebastian ; Goos, Gerhard: Copy coalescing by graph recoloring. In: *SIGPLAN Not.* 43 (2008), Nr. 6, S. 227–237. – ISSN 0362–1340

[HGG05] Hack, Sebastian ; Grund, Daniel ; Goos, Gerhard: Towards Register Allocation for Programs in SSA-form. 2005. – Forschungsbericht

[HKS03] Hirnschrott, Ulrich ; Krall, Andreas ; Scholz, Bernhard: Graph coloring vs. optimal register allocation for optimizing compilers. In: *In Joint Modular Languages Conference*, Springer Press, 2003, S. 202–213

[HS06] Hames, Lang ; Scholz, Bernhard: Nearly optimal register allocation with PBQP. In: *In Proceedings of the 7th Joint Modular Languages Conference (JMLC'06). LNCS*, Springer, 2006, S. 346–361

[Lin02] Lindenmaier, Götz: libFIRM – A Library for Compiler Optimization Research Implementing FIRM. Universität Karlsruhe, Fakultät für Informatik, Sep 2002 (2002-5). – Forschungsbericht. – 75 S

[RWZ88] Rosen, B. K. ; Wegman, M. N. ; Zadeck, F. K.: Global value numbers and redundant computations. In: *POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* New York, NY, USA : ACM, 1988. – ISBN 0–89791–252–7, S. 12–27

[SE02] Scholz, Bernhard ; Eckstein, Erik: Register allocation for irregular architectures. In: *SIGPLAN Not.* 37 (2002), Nr. 7, S. 139–148. – ISSN 0362–1340

[TLB99] Trapp, Martin ; Lindenmaier, Götz ; Boesler, Boris: Documentation of the Intermediate Representation FIRM / Universität Karlsruhe, Fakultät für Informatik. Universität Karlsruhe, Fakultät für Informatik, Dec 1999 (1999-14). – Forschungsbericht. – 0–40 S