Karlsruher Institut für Technologie

Fakultät für **Informatik**

Erstellung und Bewertung einer Bedienoberfläche zur musterbasierten Parallelisierung

Bachelorarbeit
von

Tobias Michael Müller

Verantwortlicher Betreuer:	Prof. Dr. Walter F. Tichy
Betreuender Mitarbeiter:	Dipl.-Inform. Korbinian Molitorisz

Bearbeitungszeit: 15. Mai 2014 – 15. September 2014

Karlsruher Institut für Technologie · Institut für Programmstrukturen und Datenorganisation ·
Geb. 50.34 · Am Fasanengarten 5 · 76131 Karlsruhe

Ehrenwörtliche Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Karlsruhe, den 15. September 2014

Tobias Michael Müller

Inhaltsverzeichnis

1 EINLEITUNG

Computer haben ein grundsätzlichen Problem: sie sind stets zu langsam.

Immer weiter steigender technischer Fortschritt verlangt auch steigende Rechenleistung. Wie der in Abbildung 1 dargestellte Verlauf der Prozessorentwicklung der letzten Jahre jedoch zeigt, steigt seit Beginn des 21. Jahrhunderts die Prozessorleistung immer weniger stark an. Der Grund hierfür ist, dass der Prozessortechnik physikalische und wirtschaftliche Grenzen, wie beispielsweise die enorme Wärmeentwicklung, gesetzt sind. Bereits in wenigen Jahren werden diese Grenzen endgültig erreicht werden [FM11].

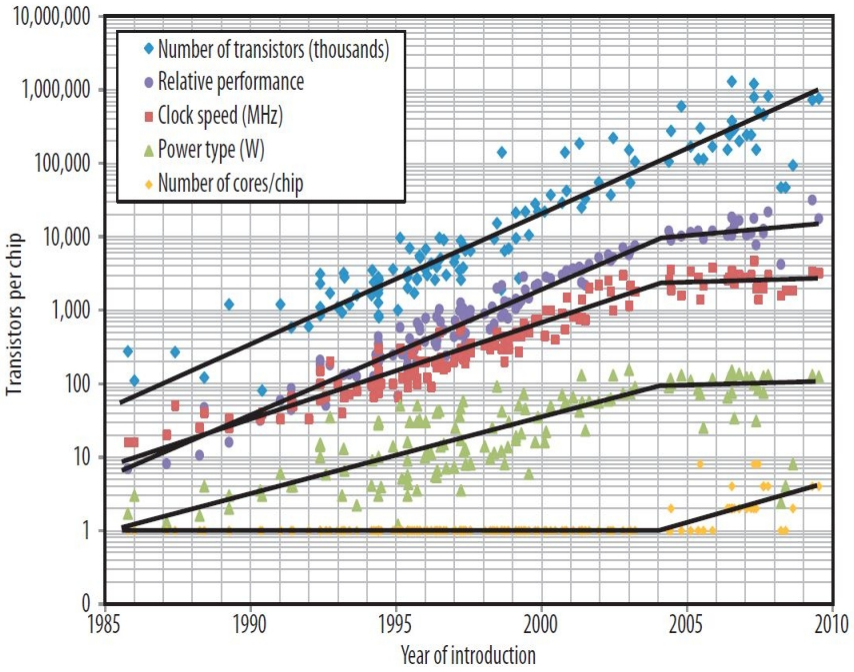

Abbildung 1: Zeitlicher Verlauf der Prozessorentwicklung (aus [FM11])

Um dem wachsenden Bedarf an Rechenleistung decken zu können, ist der bisherige Ansatz, Prozessoren immer leistungsfähiger zu machen, aufgrund der eben genannten Grenzen keine mögliche Lösung. Es ist also notwendig, hier eine Alternative zu finden. Die Hardwareindustrie ist deshalb bereits seit mehreren Jahren auf Mehrkernprozessoren umgestiegen. Anstatt den gesamten Rechenaufwand einer Einheit zuzuteilen, welche so immer mehr leisten muss, wird eine Leistungssteigerung seit 2004/2005 hauptsächlich durch das Aufteilen der Berechnungen auf mehrere Einheiten erreicht [FM11].

Damit diese Hardwaresysteme optimal ausgenutzt werden, müssen aber auch Softwaresysteme die Möglichkeit zur nebenläufigen Ausführung bieten. Das Entwickeln paralleler Programme ist jedoch schwerer, fehleranfälliger und zeitaufwändiger im Vergleich zur „herkömmlichen“ seriellen Entwicklung. Zudem gibt es große Mengen an älteren Softwaresystemen, welche nicht für Mehrkernprozessoren entwickelt wurden, da dieses Konzept zu dieser Zeit noch nicht bekannt war. Sollen diese an die neuen Systeme angepasst werden, so müssen sie bisher aufwändig manuell parallelisiert werden. Dies hat zur Folge, dass Personal gebunden wird und höhere Kosten auftreten. Eine Automatisierung des Vorgangs kann hier eine Verbesserung herbeiführen.

Wie die Schnittstelle zu einer solchen Automatisierung gestaltet werden soll und ob sich das Entwickeln solcher Werkzeugen tatsächlich lohnt, soll in dieser Arbeit ermittelt werden.

2 GRUNDLAGEN

In diesem Kapitel werden die für das Verständnis der Arbeit notwendigen Grundlagen erläutert. Erklärt werden zunächst in der Arbeit vorkommende Begriffe. Anschließend wird auf Abhängigkeiten in Programmen und Parallelisierungsstrategien eingegangen

2.1 Begriffserklärungen

Source-to-Source Compiler

Aufgabe eines Übersetzers (*engl. Compiler*) ist das Übersetzten eines Programms in eine andere Programmiersprache. Meistens wird von einem menschenlesbaren Programmquelltext (*engl. source code*) in eine auf dem Computer ausführbare Maschinensprache übersetzt. Der Code wird also von einer höheren Abstraktionsebene auf eine maschinennähere gebracht. Bei einem *Source-to-Source* Compiler hingegen wird der Code auf ungefähr derselben Abstraktionsebene übersetzt. Auch muss er nicht zwangsläufig in eine von der ursprünglichen verschiedene Programmiersprache übersetzt werden [STS].

TADL

TADL (engl. *Tunable Architecture Description Language*) ist eine Architekturbeschreibungssprache. Sie dient der expliziten Spezifikation von parallelen Architekturen und abstrahiert von der konkreten Implementierung. Eine genauere Definition inklusive der Operatoren findet sich in [W13].

Benutzerschnittstelle

Die Definition der DIN EN ISO 9241-110 vereint unter dem Begriff Benutzungsschnittstelle alle Bestandteile eines interaktiven Systems (sowohl Software als auch Hardware), die dem Benutzer notwendige Informationen und Steuerelemente zur Ausführung bestimmter Arbeitsaufgaben mit dem System zur Verfügung stellen. Die grafische Benutzeroberfläche (engl *graphical user interface/GUI*) ist somit nur ein Teil der gesamten Benutzungsschnittstelle (aus [SB11]).

Mittelwert

Der Begriff Mittelwert wird in dieser Arbeit synonym für das arithmetische oder auch Stichproben-Mittel verwendet. Es berechnet sich aus der Summe aller Messwerte geteilt durch die Anzahl an Messwerten [HK10].

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Stichproben-Varianz und Stichproben-Standardabweichung

Diese beiden Werte sind ein Maß für die Streuung von Ergebnissen. Sie zeigen also an, ob die gesammelten Messwerte eng beieinander liegen oder sich über den Messbereich verteilen. Die Varianz berechnet sich wie folgt [HK10]:

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Die Stichprobenstandardabweichung ist die Quadratwurzel der Stichproben-Varianz: $s_x = \sqrt{s_x^2}$

2.2 Abhängigkeiten

Befehle, aus denen ein Programm aufgebaut wird, beeinflussen sich gegenseitig. Durch diese Beeinflussungen entstehen Abhängigkeiten innerhalb eines Programms. Man unterscheidet im Wesentlichen zwischen zwei verschiedenen Arten von Abhängigkeit: Kontrollfluss- und Datenabhängigkeiten.

2.2.1 Kontrollflussabhängigkeit

Eine Anweisung ist von einer anderen Anweisung kontrollflussabhängig, wenn die Auswertung der vorangehenden Anweisung die Ausführung der folgenden bedingt. Das einfachste Beispiel ist ein `if-else`-Block. Die jeweiligen Blöcke sind von der Auswertung der `if`-Anweisung kontrollflussabhängig, da die Auswertung der Bedingung entscheidet, ob sie ausgeführt werden oder nicht.

2.2.2 Datenabhängigkeit

Datenabhängigkeiten treten auf, wenn mehrere Programmbefehle auf denselben Daten arbeiten. Man unterteilt sie weiter in folgende drei Kategorien [RO12]:

1. Echte Datenabhängigkeit
2. Gegenabhängigkeit
3. Ausgabeabhängigkeit

Echte Datenabhängigkeit

Eine echte Datenabhängigkeit (engl. *true dependece* oder *read-after-write*) liegt vor, wenn eine Anweisung ein Datum liest, welches zuvor von einer anderen geschrieben wird.
Beispiel Datenabhängigkeit in „a“:

```
a = 1
d = a + e
```

Gegenabhängigkeit

Eine Anweisung heißt von einer anderen gegenabhängig (*engl. anti-dependent, write-after-read*), wenn sie ein Datum verändert, welches davor ausgelesen wird.
Beispiel Gegenabhängigkeit in „b“:

```
a = b + c
b = 5
```

Ausgabeabhängigkeit

Eine Ausgabeabhängigkeit (*engl. output dependence, write-after-write*) tritt dann auf, wenn zwei Anweisungen dasselbe Datum verändern.
Beispiel Ausgabeabhängigkeit in „a“:

```
a = 42
a = b + 12
```

Bei serieller Befehlsausführung stellen Datenabhängigkeiten kein Problem dar. Lässt man die Befehle jedoch parallel laufen, so können sie zu Datenkonflikten führen, welche die Berechnung des Programms verfälschen. Datenabhängigkeiten sind jedoch nur notwendige, aber keine hinreichende Bedingung für einen Datenkonflikt.

2.2.3 Datenkonflikte

Datenkonflikte können als Resultat einer Datenabhängigkeit auftreten. Man unterscheidet Datenkonflikte anhand der Abhängigkeit, welche sie ausgelöst hat.

Eine echte Abhängigkeit führt zum Konflikt, wenn der folgende Befehl auf die Daten bereits lesend zugreift, bevor sie endgültig von vorhergehenden geschrieben wurden. Es werden dann veraltete und eventuell ungültige Daten ausgelesen, welche zu einem falschen Ergebnis führen.

Gegenabhängigkeiten führen dann zu einem Konflikt, wenn das Schreiben der Daten abgeschlossen wird, bevor der vorhergehende Befehl diese ausgelesen hat.

Konflikte werden durch Ausgabeabhängigkeiten dann ausgelöst, wenn der nachfolgende Befehl sein Ergebnis zuerst abspeichert und somit vom eigentlich vorhergehenden überschrieben wird.

2.3 Parallelisierungsstrategien

Um ein Programm auf Mehrkernrechnern ausführen zu können, muss es für diese Ausführung angepasst werden. Hierfür gibt es verschiedene Ansätze, wovon für diese Arbeit das Pipelinemuster und die Master/Worker-Architektur interessant sind.

2.3.1 Pipelineparallelisierung

Bei der Pipelineparallelisierung wird das Arbeiten auf den Programmdaten des Programms in Stufen aufgeteilt. Die Stufen arbeiten mit den Daten, welche sie von der vorhergehenden Stufe erhalten. Die Pipelinestufen können aufgrund von Abhängigkeiten nicht parallel ausgeführt werden. Die nebenläufige Ausführung entsteht dadurch, dass eine Stufe nach Weitergabe ihrer Ergebnisse direkt mit dem nächsten Datensatz weiterarbeitet. Hierfür hat jede Stufe ihren eigenen Programmfaden (engl. *thread*).

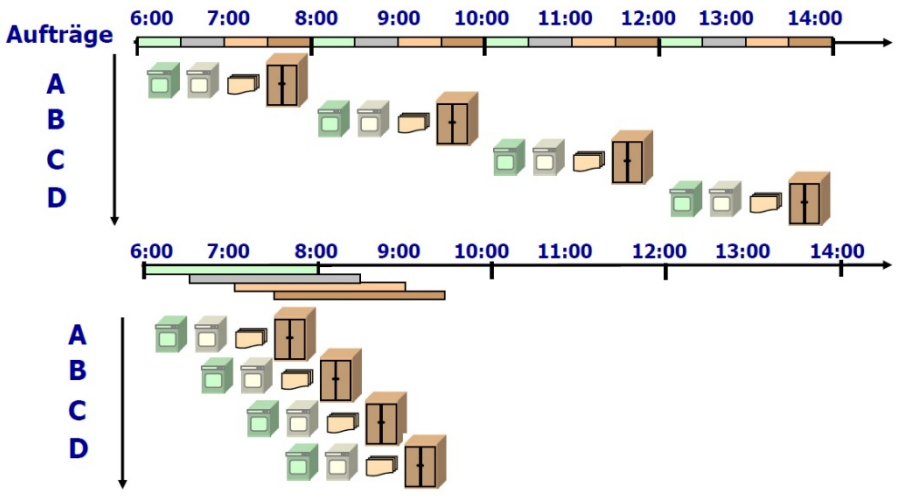

Abbildung 2: Beispielhafte Umsetzung des Pipelineverfahrens anhand eines alltäglichen Beispiels

Beispiel (aus [RO12]): Ein Waschvorgang kann in vier Teilvorgänge unterteilt werden

- Wäsche in die Waschmaschine legen
- Wäsche in den Trockner legen
- Bügeln
- Einräumen in den Kleiderschrank

Der Geschwindigkeitsvorteil des Pipelineverfahrens wird in Abbildung 2 visualisiert. Der obere Teil gibt die Dauer einer seriellen Ausführung an. Unten sieht man das Ergebnis der Anwendung des Pipelineverfahrens.

2.3.2 Master/Worker-Parallelisierung

Beim *Master/Worker* (deutsch: Arbeitgeber/Arbeitnehmer)-Architekturmuster werden voneinander unabhängige Teilaufgaben parallel ausgeführt. *Master/Worker* ist eine Realisierung des „Teile und Herrsche“-Prinzips. Hierfür stellt eine Instanz, der sogenannte *Master*, die zu erledigenden Aufgaben bereit und verteilt sie an die *Worker*, welche diese dann unabhängig von den anderen ausführen. Die Ergebnisse der Berechnungen werden vom Master gesammelt. Sind alle vorhanden, so wird die Programmausführung fortgeführt [ES13].

3 VERWANDTE ARBEITEN

Im Folgenden werden Arbeiten betrachtet und verglichen, die sich ebenfalls damit beschäftigen, seriellen Quellcode zu Parallelisieren und dem Benutzer dies zu erleichtern. Betrachtet werden ParaGraph anhand von [BF10a][BF10b], HTGviz mittels [GF99], Prism [CRI] sowie das Parallel Studio Letzteres wurde anhand einer installierten Testversion betrachtet. Es werden die Stärken und Schwächen der Werkzeuge analysiert, um daraus Erkenntnisse zur eigenen Konzeption zu erhalten. ParaGraph und HTGviz sind einfache Werkzeuge und beschäftigen sich im Wesentlichen nur mit der Visualisierung von Abhängigkeiten anhand von Daten, die sie aus einer externen Quelle beziehen. Prism und Parallel Studio als kommerzielle Produkte dagegen sind deutlich umfangreicher und bieten unter anderem eigene Übersetzter (engl. *Compiler*) sowie zahlreiche Analysewerkzeuge.

3.1 ParaGraph

ParaGraph ist ein in [BF10a] und [BF10b] vorgestelltes Werkzeug, welches dem Benutzer bei der Parallelisierung von C-Quellcode unterstützen soll. ParaGraph wurde als *Plug-in* für die Entwicklungsumgebung (engl. *integrated development environment/ IDE*) Eclipse entwickelt und ist somit plattformunabhängig.
Die Kernfunktion von ParaGraph ist die Visualisierung des Kontrollflussgraphen mit dem Ziel, Datenabhängigkeiten für den Entwickler offenzulegen, um so den Parallelisierungsprozess nachvollziehbarer zu gestalten.

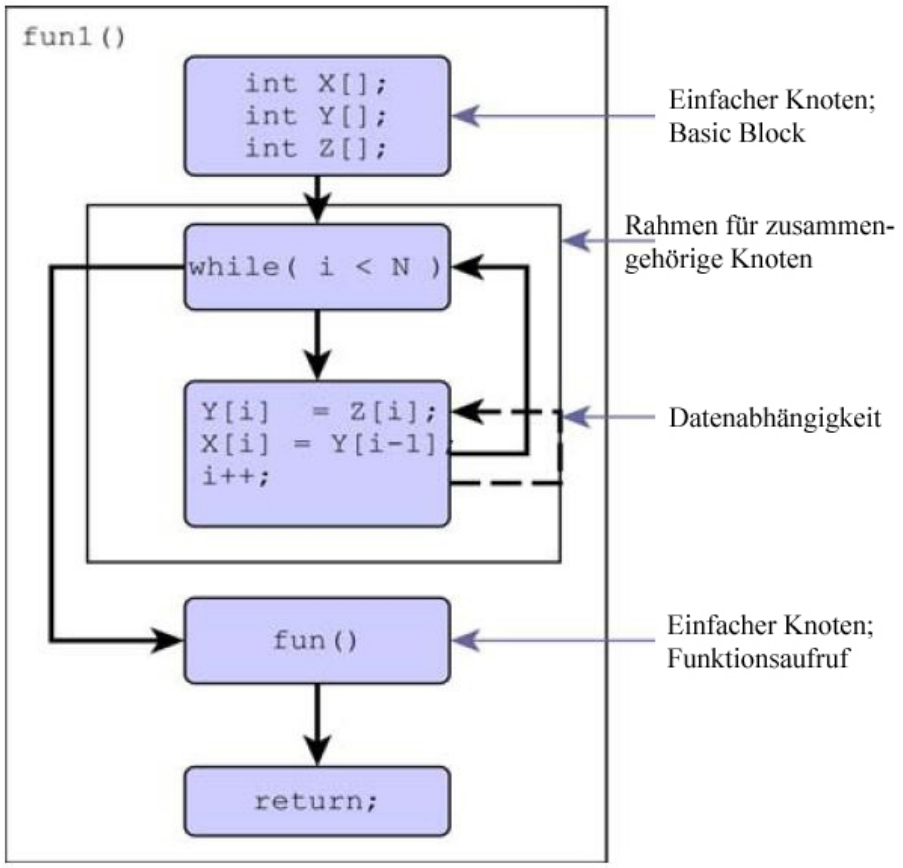

Abbildung 3: Darstellung des Abhängigkeitsgraphen in ParaGraph.

ParaGraph ist dabei jedoch nicht selbstständig, sondern auf einen externen *Source-to-Source* Compiler angewiesen (z.B. Cetus[BB+09]). Insbesondere muss dieser Compiler die Kontrollfluss- und Datenabhängigkeiten erkennen.

Die darauf aufbauende Visualisierung des Graphen, welche in Abbildung 3 zu sehen ist, erfolgt dann durch ParaGraph wie folgt:
Als Knoten werden zusammengehörige Programmabschnitte, wie zum Beispiel der Rumpf einer Schleife verwendet. Die Knoten werden mit den jeweiligen Quellcodeinstruktionen beschriftet. Zusammengehörige Knoten, wie beispielsweise Kopf und Rumpf einer Schleife, werden durch einen Rahmen visuell gekennzeichnet. Für den Graphen gibt es zwei verschiedene Kantenrelationen

1. die Vorgängerrelation, um den Kontrollfluss darzustellen. Darstellung mittels durchgezogener Pfeile.
2. die Datenabhängigkeiten, um dem Benutzer die Stellen, welche eine Parallelisierung verhindern, anzuzeigen. Darstellung mittels gestichelter Pfeile.

Schwachpunkt der Darstellung des Graphen ist die fehlende Unterscheidung der Datenabhängigkeiten zwischen echten, Gegen- und Ausgabeabhängigkeiten. Der Abhängigkeitsgraph ist nicht editierbar, Änderungen werden immer direkt im Quellcode vorgenommen. Er wird in einem separaten Register dargestellt. Dies führt dazu, dass der direkte Bezug zum Quellcode fehlt und somit immer zwischen den verschiedenen Registern gewechselt werden muss. Durch Klicken im Graphen kann man allerdings direkt zur entsprechenden Quellcodestelle gelangen.

3.2 HTGviz

HTGviz steht für *Hirarchical Task Graph Visualization Tool* und wird in [GF99] vorgestellt. Wie ParaGraph benötigt auch HTGviz einen externen Compiler (z.B. Parafrase-2).

HTGviz ist nicht in eine Entwicklungsumgebung eingebunden, sondern baut auf dem Compiler auf und stellt eine grafische Benutzeroberfläche (engl. *graphical user interface/ GUI*) für den Übersetzungsprozess dar. Hauptfunktion ist auch hier die Abbildung des Kontrollflussgraphen mit Datenabhängigkeiten. Im Gegenteil zu ParaGraph kann hier der Graph direkt bearbeitet werden. Der Benutzer kann Knoten zusammenfügen, aufteilen, hinzufügen und entfernen, um so die Abhängigkeiten, welche der Parallelisierung im Weg stehen, zu beheben.

Der Programmquelltext wird in sogenannte *Tasks* aufgeteilt. Tasks sind Codesegmente, welche sich durch „natürliche“ Grenzen (Schleifenrümpfe, Basisblöcke (engl. *basic block*), Methodenaufrufe und Anweisungen) definieren. Die Darstellung des HTG kann sehr komplex werden, da beispielsweise Schleifen verschachtelt vorkommen können. Da eine Schleife immer einen *Task* repräsentiert, können so also innerhalb eines *Tasks* weitere *Tasks*, sogenannte *Subtasks*, enthalten sein. Um durch die so entstehenden Ebenen zu navigieren, können Knoten ausgeklappt oder zur besseren Übersicht zusammengefasst werden. Außerdem werden auch hier zusammengehörige Knoten durch Rahmen entsprechend markiert.

Die Darstellung des HTG ist ähnlich der Darstellung in ParaGraph. Als Knoten werden die einzelnen *Tasks* verwendet. Es gibt hier jedoch drei Kantenrelationen, welche in der grafischen Darstellung auch jeweils verborgen werden können

1. die Vorgängerrelation, um den Kontrollfluss darzustellen. Darstellung mittels durchgezogener Pfeile.
2. die Datenabhängigkeiten. Darstellung mittels durchgezogenen Pfeilen versehen mit Strichen.

3. die Kontrollflussabhängigkeiten. Dargestellt mittels durchgezogenen Pfeilen versehen mit Punkten.

Zu dem Graphen bietet das Programm ein Informationsfenster zu den Knoten und Kanten des Graphen. Hier wird insbesondere die bei Paragraph noch fehlende Unterscheidung der Abhängigkeiten getroffen. Außerdem werden die verschiedenen Katentypen farblich hervorgehoben. Da HTGviz nicht in eine IDE eingebunden ist, muss es zusätzlich einen eigenen Editor bereitstellen.

3.3 Prism

Prism ist ein in die Entwicklungsumgebung Eclipse eingebundenes Werkzeug der Firma CriticalBlue [CRI], welches zum Analysieren von Software hinsichtlich möglicher Parallelisierung verwendet wird. Es nimmt nicht selbstständig Änderungen am Quellcode vor, sondern hilft dem Benutzer, Stellen zu finden, welche die Parallelisierung verhindern. Weiter simuliert Prism, wie das Programm abläuft, wenn der Benutzer bestimmte Stellen parallelisieren würde. Zudem ist es auch möglich, die Ausführung auf einer beliebigen Anzahl von Kernen zu simulieren.
Das Arbeiten mit Prism teilt sich im Wesentlichen in fünf Schritte auf:

1. ***Analyze*/Analyse**
   Prism führt den Quellcode aus, ermittelt die Laufzeiten der Codeabschnitte und erkennt Abhängigkeiten. Prism zeigt dem Benutzer die bei der Analyse gemessenen Laufzeiten je Codeabschnitt in Prozent an der gesamten Laufzeit. Weiter wird die Laufzeit per Balkendiagramm dargestellt. So werden Stellen, die mögliches Parallelisierungspotential bieten, identifiziert.
2. ***Explore*/Erforschen**
   Der Benutzer simuliert verschiedene Szenarien (*What-if*-Fragen; Was passiert, wenn ich hier parallelisiere/diese Abhängigkeit entferne/…). Er erhält von Prism eine grafische Darstellung des simulierten Programmablaufs.
3. ***Implement*/Implementierung**
   Der Benutzer wählt anhand der gewonnen Ergebnisse eine Parallelisierungsstrategie und schreibt den Code in der *IDE* um. Prism selbst nimmt keine Änderungen am Quelltext vor, sondern verlässt sich hier auf die Entscheidungen des Entwicklers.
4. ***Verify*/Verifizierung.**
   Durch die Änderungen können neue Abhängigkeiten entstehen. Prism prüft, ob die Parallelisierung problemlos abläuft oder ob im veränderten Programm neue Wettläufe entstehen.
5. ***Tune*/Anpassen**
   Den Vorgang wiederholen, um die Parallelität noch weiter zu erhöhen und die Laufzeit zu optimieren.

Für diese Arbeit sind besonders die Schritte eins bis drei interessant. Die Ergebnisse einer Simulation in Prism werden durch eine Art Balkendiagramm präsentiert. Auf der Y-Achse sind die Kerne angegeben, die X-Achse ist die Zeitachse. Wenn eine zentrale Recheneinheit (engl. *central processing unit/CPU)* beschäftigt ist, so wird dies durch einen Balken visualisiert. Datenabhängigkeiten werden, im Gegensatz zu ParaGraph und HTGviz, nicht anhand eines Kontrollflussgraphen dargestellt. Sie werden dagegen durch Pfeile zwischen den einzelnen Blöcken angegeben. Durch Klicken auf die Pfeile gelangt man im Quellcode zur entsprechenden Codestelle. Diese Visualisierung ist weniger übersichtlich als die Darstellung im Graphen, insbesondere, da die Balken nicht beschriftet sind und so jeder Bezug fehlt.

3.4 Parallel Studio

Parallel Studio ist eine von Intel entwickelte Software, welche es dem Benutzer erleichtern soll, seriellen Quellcode für die Ausführung auf mehreren Kernen anzupassen. Parallel Studio bietet hierfür verschiedene Werkzeuge:

1. ***VTune Amplifier***
   Mit diesem Werkzeug kann die Performanz paralleler Anwendungen ermittelt werden. Es werden drei verschiedene Analysearten angeboten, die jeweils verschiedene Fragestellungen beantworten
   a. *Hotspots* – Wofür benötigt der Code die meiste Laufzeit?
   b. *Concurrency* – Wie gut skaliert der Code auf mehreren Kernen?
   c. *Looks and Waits* – Wo verbraucht die Synchronisation die meiste Zeit?

   Dem Benutzer werden nach der Analyse zahlreiche Daten präsentiert:
   - Die Gesamtlaufzeit
   - Wie lange war die *CPU* beschäftigt/untätig
   - Anzahl der Kerne
   - Anzahl der Fäden
   - Wie lange/oft mussten Fäden warten
   - Methoden/Fäden, absteigend nach Laufzeit sortiert. Die Laufzeit wird sowohl in Sekunden, als auch durch einen Balken dargestellt

   Durch Klicken auf den Methodennamen gelangt man direkt in den Quellcode. Die einzelnen Codezeilen sind ebenfalls mit der Rechenzeit in Sekunden sowie einem Balken versehen. Bei der Hotspotanalyse sind die Balken einfarbig, bei den beiden anderen wird durch Farben angezeigt, wie gut die *CPU* ausgenutzt wurde (z.B. bedeutet grün „ideal“, rot eine schlechte Auslastung).

2. ***Parallel Inspector***
   Parallel Inspector wird benutzt, um Programme auf *Threading-* und *Memory-Errors* zu untersuchen, welche bei der Parallelisierung entstehen können. Wird beispielsweise ein Datenwettlauf entdeckt, so kann wieder direkt in die entsprechende Codezeile gesprungen werden.

3. ***Parallel Advisor***
   Hier kann ein Entwickler ermittelt, wo mit dem Parallelisieren begonnen werden soll und ob eine Parallelisierung sinnvoll ist. Zunächst wird hierfür eine Laufzeitanalyse durchgeführt. Dem Benutzer wird für jeden Programmabschnitt aufgezeigt, wieviel Rechenzeit dort verwendet wurde. Die Auswertung kann entweder Tabellarisch oder, wie in Abbildung 4 zu sehen, in direktem Bezug zum Quelltext eingesehen werden. Es Aufgabe des Entwicklers, nun Quelltextstellen, welche er für sinnvoll hält, zu annotieren.

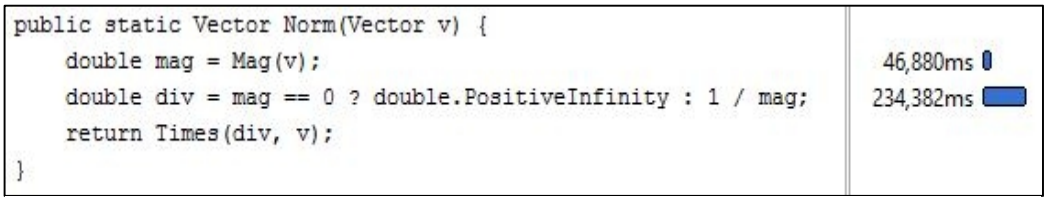

Abbildung 4: Bildschirmausschnitt aus Parallel Studio. Ergebnis der Laufzeitanalyse im Bezug zum Quelltext

Diese Entscheidung trifft er nur Aufgrund der es von Parallel Studio errechneten laufzeitverhaltens. Abhängigkeiten zwischen den Codesegmenten werden dem Benutzer nicht angeboten. Für die Annotation steht dem Anwender bei Bedarf ein Assistent zur Verfügung, welcher die Annotationen umsetzt. Anschließend kann sich der Anwender eine Laufzeitvorhersage ermitteln lassen, um zu entscheiden, ob sich die Parallelisierung hier lohnt. Parallel Studio betrachtet hier nur die von Benutzer annotierten Stellen.

Parallel Studio ist als kommerzielles Produkt das umfangreichste der betrachteten Werkzeuge. Es bietet die meisten und detailliertesten Informationen und Analysemöglichkeiten. Allerdings kann ein großer Teil das Potential der vielen Analysemöglichkeiten und sonstigen Unterstützungen verloren gehen, da sich Parallel Studio im Verlauf der Parallelisierung auf Entscheidungen des Entwicklers verlässt.

3.5 Fazit

ParaGraph und HTGviz bieten hauptsächlich nur die Möglichkeit, Abhängigkeiten hervorzuheben und zeigen dem Benutzer so auf, welche Codeabschnitte eine Parallelisierung verhindern. Prism und Parallel Studio als kommerzielle Produkte dagegen bieten noch deutlich mehr Funktionen, wie beispielsweise Simulationen der Skalierbarkeit. Aus den vier hier betrachteten Werkzeugen sollen folgende Aspekte übernommen werden:

- **Einbindung in eine Entwicklungsumgebung**
  Die Einbindung in eine bekannte Entwicklungsumgebung bringt mehrere Vorteile mit sich. Da beispielsweise Visual Studio sehr bekannt ist, bleibt der Benutzer in seiner gewohnten Umgebung und kann so schneller lernen, mit dem Programm umzugehen. Weiter muss man keinen Editor oder andere Werkzeuge entwickeln.
- **Darstellung der Abhängigkeiten in einem Abhängigkeitsgraphen**
  Die Darstellung als Graph ist deutlich leichter verständlich als beispielsweise die tabellarische Auflistung im Parallel Studio. Allerdings kann sie zu sehr komplexen Graphen führen.
- **Verbindung zwischen der Darstellung der Abhängigkeiten und dem zugrundeliegenden Quelltext**
  Alle Werkzeuge bieten die Möglichkeit, anhand ihrer Visualisierung direkt in die entsprechende Codezeile zu springen. Dies erlaubt dem Benutzer, alle Vorgänge einfach nachvollziehen zu können, und erspart langes Suchen.
- **Visualisierung des Laufzeitverhaltens**
  Eine Erstellung des Laufzeitprofils hilft dem Anwender zu verstehen, an welchen Stellen das Programm viel Rechenaufwand betreibt und so für eine Parallelisierung in Frage kommt.
  Codesequenzen, welche zwar parallelisierbar sind, eine Parallelisierung jedoch keinen Laufzeitvorteil oder sogar einen Nachteil mit sich bringen würde, können so auch teilweise erkannt werden.

Abbildung 5 fasst die positiven Merkmale sowie die Defizite der in diesem Kapitel betrachteten Werkzeuge noch einmal tabellarisch zusammen.

Werkzeug	Lösungen	Positive Merkmale	Defizite
ParaGraph	• Eingebunden in IDE (Eclipse) • Visualisierung des Kontrollflussgraphen • Annotieren des Quelltextes	• Abhängigkeitsgraph nicht editierbar • Direktes Springen vom Graphen in Quelltext • Eigene Annotationen möglich	• Auf externen Compiler zur Analyse angewiesen • Direkter Bezug zum Quelltext fehlt, da Darstellung des Graphen in eigenem Register
HTGviz	• Eigenständige Anwendung • Visualisierung des Kontrollflussgraphen	• Ausklappen und zusammenführen von Knoten im Graphen • Detaillierte Informationen zu einzelnen *Tasks* • Annotationsassistent	• Auf externen Compiler zur Analyse angewiesen • Direkte Manipulation des Graphen • Keine selbstständige Annotation
Prism	• Eingebunden in IDE (Eclipse) • Analyse des Quelltexts • Simulation von Parallelisierungsszenarien	• Simulation der Skalierbarkeit • Anzeige der Laufzeit eines Codeabschnitts in Prozent sowie als Balkendiagramm • Direktes Springen in Quellcodeabschnitte	• Keine selbstständigen Änderungen am Quelltext • Verlässt sich auf Entscheidungen des Entwicklers • Kein Abhängigkeitsgraph
Parallel Studio	• Eingebunden in IDE (Visual Studio) • Analyse des Quelltexts • Simulation von Parallelisierungsszenarien	• Simulation der Skalierbarkeit • Anzeige der Laufzeit eines Codeabschnitts in Prozent sowie als Balkendiagramm • Direktes Springen in Quellcodeabschnitte • Annotationsassistent	• Keine selbstständigen Änderungen am Quelltext • Verlässt sich auf Entscheidungen des Entwicklers • Kein Abhängigkeitsgraph

Abbildung 5: Zusammenfassung der betrachteten verwandten Arbeiten

4 KONZEPTION

Ziel dieser Arbeit ist das Erstellen und Bewerten einer möglichst benutzerfreundlichen Schnittstelle zur Verwendung eines mustergestützten, automatischen Parallelisierungsvorgangs. Grundlage der Benutzerschnittstelle ist das von der Forschungsgruppe AParT entwickelte Parallelisierungswerkzeug Patty. Im Folgenden sollen mögliche Herangehensweisen diskutiert werden. Zunächst wird die Zielgruppe von Patty betrachtet, sowie die an die Oberfläche gestellte Aufgabe formuliert. Darauf aufbauend werden die grundlegenden Anforderungen an die Schnittstelle definiert, gefolgt von einer genaueren Untersuchung und Diskussion von Lösungen und Umsetzungen dieser Anforderungen.

4.1 Zielgruppe

Dieses Werkzeug soll bei Benutzern Verwendung finden, die veralteten oder seriellen Quellcode an moderne Mehrkernrechner anpassen möchten. Es soll somit nicht nur von professionellen Softwareentwicklern mit Erfahrung in paralleler Programmierung verwendbar, sondern auch für unerfahrene Entwickler einfach bedienbar sein.

4.2 Aufgabenstellung

Die primäre Aufgabe von Patty ist die automatisierte Umwandlung von seriell geschriebenen Quelltexten in Code, welcher parallel ausführbar ist. Bei der Untersuchung des Quelltextes ermittelt das Werkzeug Parallelisierungspotential durch mögliche Anwendung des *Master/Worker*- oder *Pipelinevorgehenes* und transformiert anschließend die entsprechenden Codeabschnitte.

Nicht Aufgabe von Patty ist das Überprüfen des seriellen Quellcodes auf Korrektheit. Diese wird von Patty vor der Ausführung angenommen. Außerdem wird Vorausgesetzt, dass Patty der gesamte Quelltext bekannt ist. Wird dies nicht gewährleistet, indem das zu parallelisierende Programm zum Beispiel auf externe, nicht öffentliche Bibliotheken verweist, kann die Korrektheit der Ausgaben von Patty ebenfalls nicht garantiert werden.

4.2.1 Eingabedaten

Patty arbeitet im Wesentlichen in fünf Schritten (siehe Abbildung 6), welche in die beiden getrennte Teilaspekte „Quellcodeanalyse zur Identifizierung möglicher Parallelisierungsabschnitte“ und „automatische Parallelisierung von annotiertem Code“ geteilt werden können.
Pattys Programmschritte sind:

1. Statische Analyse
2. Dynamische Analyse
3. Erkennung von geeigneten Zielmustern (sogenanntes p*attern matching*)
4. *TADL*-Annotierung

→ Quellcodeanalyse zur Identifizierung möglicher Parallelisierungsabschnitte

5. Umwandlung in parallel ausführbaren Code → automatische Parallelisierung von annotiertem Code

Zur Durchführung dieser Schritte müssen dem Benutzer folgende Eingabemöglichkeiten geboten werden:

Quellcodeanalyse zur Identifizierung möglicher Parallelisierungsabschnitte

1. Der zu untersuchende Quelltext. Dieser wird von Patty als korrekt angenommen.

2. Eingabeparameter zur Simulation des Quelltexts
3. Auswahl, mit welchen Parallelisierungsmethoden (*Pipeline* oder *Master/Worker*) Patty ausgeführt wird sowie eine Priorisierung einer der beiden, falls es zur Überschneidung kommt.
4. Auswahl zwischen Schritt für Schritt Analyse inklusive Anzeige von Zwischenergebnissen oder automatischem Volldurchlauf

Automatische Parallelisierung von annotiertem Code

Der zu parallelisierende Quelltext inklusive der *TADL*-Annotationen. Die Annotationen werden von Patty als korrekt angenommen.

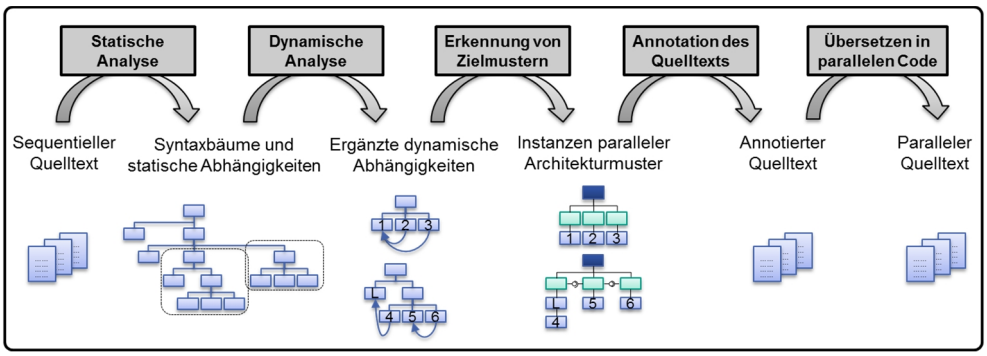

Abbildung 6: Die fünf Schritte des Parallelisierungsvorgangs von Patty inklusive der entstehenden Artefakte

4.3 Anforderungen

Der Benutzer soll die Möglichkeit haben, die beiden eben genannten (4.2.1) Teilabschnitte der Parallelisierung jeweils separat oder kombiniert in einem Durchlauf durchführen zu können. Für das Nutzungserlebnis (engl. *user experience/UX*) spielen subjektive Empfindungen der Benutzer eine große Rolle. Um eine Identifikation mit dem Parallelisierungsvorgang zu erreichen, muss dieser für den Anwender nachvollziehbar sein. Es muss jedoch darauf geachtet werden, dass bei dem Ziel, eine möglichst hohe Transparenz zu erreichen, dem Benutzer nicht zu viele und insbesondere keine für ihn sinnlosen Informationen geboten werden. An die Benutzerschnittstelle werden also folgende Anforderungen gestellt:

A1. Automatisierung des Parallelisierungsvorgangs mit flexiblen Einstiegsmöglichkeiten
A2. Nachvollziehbarkeit des Parallelisierungsvorgangs
A3. Optimum an Informationsgehalt

4.3.1 A1. Automatisierung des Parallelisierungsvorgangs mit flexiblen Einstiegsmöglichkeiten

Die Automatisierung des Parallelisierungsvorgangs ist die originäre Aufgabe von Patty. Im Gegensatz zu Prism (3.3) und Parallel Studio (3.4) soll es sich nicht ausschließlich auf Entscheidungen des Benutzers verlassen, sondern die Parallelisierung möglichst durchführen können, ohne auf Aktionen des Anwenders angewiesen zu sein. Der Benutzer soll die Möglichkeit haben, sowohl den gesamten Parallelisierungsprozess am Stück durchlaufen zu lassen, als auch nach der Analyse einzusteigen und bereits annotierten Quellcode zu übersetzen oder nur seinen Quellcode auf mögliches Parallelisierungspotential untersuchen zu lassen. Hat der Benutzer

bereits selbst seinen Quelltext auf Parallelisierungsmuster untersucht und möchte ihn übersetzen lassen, so muss er die entsprechenden Codeabschnitte mit TADL (engl. *tunable architecture description language*)-Anweisungen annotieren.

4.3.2 A2. Nachvollziehbarkeit des Parallelisierungsvorgangs

Wie eingangs erwähnt, sind die Empfindungen für die *User Experience* entscheidend. Ein undurchsichtiges Verfahren (engl. *black box*), welches den benutzergeschriebenen Quelltext verändert, ohne mitzuteilen weshalb, wird nur schwer Akzeptanz finden.

Wie wichtig Nachvollziehbarkeit tatsächlich ist, wurde bei IBM 1999 am Beispiel ihrer Webseite klar. Die meistgenutzte Funktion der Seite war die Suchfunktion, da den Benutzern nicht klar wurde, wie die Seite zu bedienen war. Die am zweithäufigsten verwendete Funktion war der Hilfe-Knopf, weil die Suchfunktion nicht effektiv war. IBM entwickelte daraufhin mit über 100 Mitarbeitern zehn Wochen lang die Seite unter enormen Kosten neu. Nachdem die Webseite überarbeitet war, sank die Benutzung des Hilfe-Knopfs um 84%, wohingegen der Umsatz um 400% anstieg [SWT13].

Um solche Vorkommnisse zu vermeiden, muss der Parallelisierungsvorgang für den Anwender nachvollziehbar gestaltet werden. Dies muss bereits bei grundsätzlichen Überlegungen wie der Art der Schnittstelle miteinbezogen werden

Kommandozeile

Kommandosprachen haben den Vorteil, dass sie sehr effizient sind [SB11], da das System mit wenig Eingaben gesteuert und die Befehle für das System optimiert werden. Allerdings sind sie nicht selbsterklärend oder ausprobierbar, sondern nur dann zu bedienen, wenn alle Befehle bekannt sind. Da Patty insbesondere auch von Benutzern, die ohnehin unerfahren auf dem Gebiet der Parallelisierung sind, bedient werden soll, ist es nicht sinnvoll, eine Kommandosprache zu verwenden.

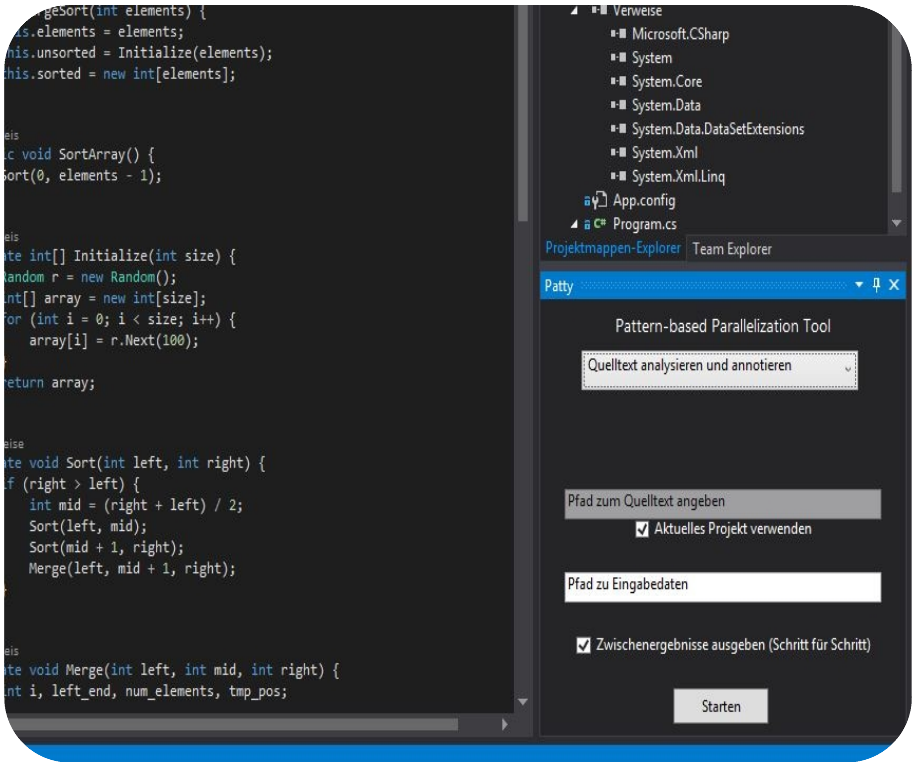

Abbildung 7: Attrappe (engl. *mock-up*) der Einbindung von Patty in VS12

Grafische Benutzeroberfläche

Eine grafische Benutzerschnittstelle (engl. *graphical user interface/GUI*) ist im Vergleich zu einer Kommandosprache hinsichtlich der Realisierung deutlich komplexer und weniger effizient, jedoch sind sie, abgesehen von professionellen Umfeldern, die bekannteste Interaktionsweise und somit sind die meisten Benutzer hiermit vertraut. Da *GUI*s die Möglichkeit bieten, Informationen bildlich darzustellen, können sie durch geeignete Symbole selbsterklärend und deutlich intuitiver als eine Kommandosprache sein. Somit wird Patty als *GUI* realisiert. Abbildung 7 zeigt eine mögliche Umsetzung einer solchen *GUI*.

Grundsätzlicher Aufbau der GUI

Bei der Erstellung einer *GUI* steht man grundsätzlich vor der Frage, ob man mehr im Bezug zu den im System gespeicherten Daten oder zu den vom Benutzer ausführbaren Arbeitsaufgaben entwickelt. Eine an die Arbeitsaufgaben angepasste Entwicklung kann sich nachteilig auf die Gebrauchstauglichkeit (engl. *usability*) auswirken [SB11]. So hat man eventuell verschiedene Eingabefenster für ähnliche Arbeitsaufgaben. Eine Arbeitsaufgabe kann viele Schritte beinhalten und so zu fehlendem Überblick der Daten führen. Da Patty jedoch keine persistenten Daten enthält und der Benutzer Schritt für Schritt durch den Parallelisierungsprozess geführt werden soll, wird sich die grafische Benutzeroberfläche nach den Arbeitsaufgaben richten.

Umsetzung

Im Kapitel Verwandte Arbeiten hat sich gezeigt, dass eine Einbindung in eine schon bestehende Entwicklungsumgebung (engl. *integrated development environment/ IDE*) sinnvoll ist. Da Patty in der Programmiersprache C# entwickelt wurde, wird es als VSPackage in das Microsoft Visual Studio eingebunden.

Darstellung

Die Darstellung der Zwischenergebnisse hilft, den Prozess nachvollziehbarer zu gestalten. Die Nachvollziehbarkeit des Prozesses soll durch die in den jeweiligen Schritten entstehenden Artefakte geschehen. In diesem Abschnitt wird die Darstellung der Zwischenergebnisse betrachtet. Welche sich hierfür eignen, wird im Abschnitt über Anforderung A3 (4.3.3) untersucht.

Parallelisierungspotential

Um dem Benutzer die von Patty durchgeführten Analysen klar zu machen, sollen ihm die Codestellen mit Parallelisierungspotential präsentiert werden. Zudem sollen auch die Codeabschnitte, welche beispielsweise aufgrund von Datenabhängigkeiten bereits verworfen wurden, einsehbar sein.

Um einen der Vorteile der Einbindung in eine IDE zu nutzen, soll diese Visualisierung direkt im Editor durch Markieren der entsprechenden Codezeilen geschehen. Der Intuition entsprechen sollen potentiell parallelisierbare Codesegmente grün, bereits verworfene rot hinterlegt werden.

Um jedoch die Benutzbarkeit für Menschen, welche an einer Rot-Grün-Schwäche leiden, nicht einzuschränken, soll es zudem einen für sie angepassten Modus geben, der andere Farben zur Darstellung verwendet.

Der Abhängigkeitsgraph

Die in der statischen und dynamischen Analyse ermittelten Abhängigkeiten sollten, ähnlich wie bei ParaGraph oder HTGviz im Kaptiel über Verwandte Arbeiten, dem Benutzer als Graph präsentiert werden. Eine mögliche Umsetzung des Graphen zeigt Abbildung 8. Als Knoten dienen die Codeabschnitte, welche zur besseren Veranschaulichung auch die Knotenbeschriftung sind.

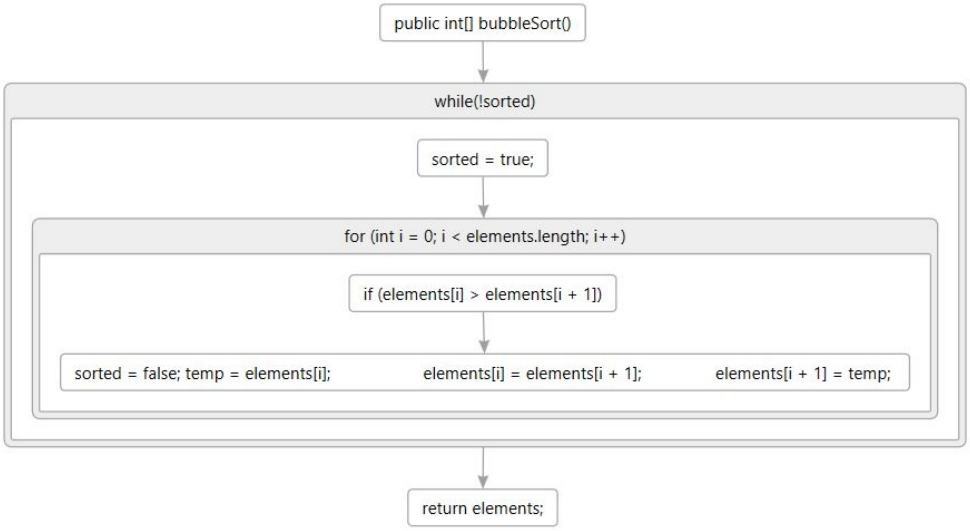

Abbildung 8: Darstellung des DGML-Kontrollflussgraphen am Beispiel von Bubblesort

Zwischen zwei Knoten können folgende Abhängigkeiten bestehen [H13]:

1. Kontrollflussabhängigkeit
2. Echte Datenabhängigkeit (engl. *true dependence, read-after-write*)
3. Gegenabhängigkeit (engl. *anti-dependence, write-after-read*)
4. Ausgabeabhängigkeit (engl. *output dependence, write-after-write*)
5. Iterationsübergreifende Abhängigkeit
6. Transitive Abhängigkeit
7. Keine Abhängigkeit

Für die Parallelisierung unwichtige Datenabhängigkeiten, (zwei aufeinanderfolgende Lesezugriffe), werden direkt verworfen und nicht visualisiert. Schleifen können verschachtelt vorkommen. Somit hat der Abhängigkeitsgraph keine reine Baumstruktur, sondern kann von einzelnen Knoten ausgehende Zyklen enthalten. Um auch hier eine möglichst übersichtliche Darstellung zu gewährleisten, sollen diese Zyklen ein- beziehungsweise ausgeklappt gestaltet werden. Die XML-basierte Beschreibungssprache DGML (*engl. directed graph markup language*) [DGML], soll zur Realisierung der Graphen verwendet werden, da sich DGML-Graphen direkt in Visual Studio öffnen lassen und so das Werkzeug in seiner Umsetzung als *Plug-in* konsequent bleibt. Beispielhaft wird in Abbildung 8 der Kontrollflussgraph von *Bubblesort*, allerdings ohne Datenabhängigkeiten, dargestellt. Um den Bezug dieser Graphen zum ursprünglichen Quelltext herzustellen, soll es möglich sein, sich aus dem Editor heraus an der zugehörigen Codestelle direkt den entsprechenden Abhängigkeitsgraphen anzeigen zu lassen.

Der mit *TADL* annotierte Quelltext soll dem Benutzer in gewohnter Art im Editor der Entwicklungsumgebung zur Verfügung stehen. Um die Zielmuster nicht nur anhand der eventuell nicht bekannten Annotationssyntax erkennen zu können, sollen sie im Editor zusätzlich textuell markiert werden.

4.3.3 A3. Optimum an Informationsgehalt

Während der Parallelisierung entstehen in den einzelnen Schritten, wie in Abbildung 6 zu sehen, verschiedene Artefakte. Damit Patty nachvollziehbar ist, den Benutzer aber dennoch nicht mit unnötigen Informationen überfordert, muss ermittelt werden, welche dieser Artefakte als geeignete Zwischenergebnisse eingesetzt werden können.

Quellcodeanalyse zur Identifizierung möglicher Parallelisierungsabschnitte

In einem der ersten Schritte ermittelt Patty alle Schleifenvorkommen im Quelltext. Dieses Artefakt dem Benutzer als Zwischenergebnis zu präsentieren ist nicht sinnvoll, da es für den Benutzer keinerlei neuer Information beinhaltet.

Bei der folgenden statischen und dynamischen Analyse ermittelt Patty Syntaxbäume der Programmmethoden sowie Datenabhängigkeiten im Quelltext. Um dem Anwender klar zu machen, welche Codeabschnitte eine Parallelisierung verhindern, ist die Visualisierung der Abhängigkeitsgraphen sehr gut geeignet.

Im auf die Analysen folgenden Schritt werden nun die ermittelten Daten mit einem Katalog von Parallelisierungsmustern verglichen und geeignete Zielmuster (*Pipelining, Master/Worker*) für jeden identifizierten Codeabschnitt bestimmt. Die bei dieser Untersuchung gefundenen Schleifen sollen dem Benutzer jedoch nicht in einem extra Schritt präsentiert, sondern erst zusammen mit den TADL-Annotationen visualisiert werden.

Die *TADL*-Annotationen bilden den Abschluss der Quellcodeanalyse. Der Anwender kann an ihnen nachvollziehen, welche Codeabschnitte seines Quelltexts sich letztendlich für eine Parallelisierung eignet. Sie sollen daher auf jeden Fall visualisiert werden.

Automatische Parallelisierung von annotiertem Code

Nach der Parallelisierung steht dem Benutzer der durch Patty parallelisierte Programmquelltext zur Verfügung. Der ausgegebene Programmtext ist im Moment jedoch kaum zu verstehen. Der Anwender hat hierdurch also keinen Mehrwert. Diese Option anzugeben ist im Hinblick auf die Zukunft dennoch sinnvoll, da davon auszugehen ist, dass dieser Teil von Patty noch überarbeitet werden wird.

4.4 Fazit

Grundsätzliche Aufgabe des Werkzeuges ist die Automatisierung des Parallelisierungsvorgangs, um es insbesondere auch unerfahrenen Entwicklern zu ermöglichen, seriellen Quelltext an moderne Prozessoren anzupassen. Oberstes Ziel bei der Entwicklung der Schnittstelle soll die Nachvollziehbarkeit sein. Patty erhält deshalb eine grafische Benutzeroberfläche. In der GUI werden auch die im Verlauf der Parallelisierung entstehenden Artefakte visualisiert. Als am wichtigsten werden hier die Abhängigkeitsgraphen erachtet. Ein gutes Maß zwischen zu vielen und zu wenigen Informationen ist entscheidend. Zu viele Informationen führen schnell dazu, Anmerkungen nichtmehr zu beachten. So können wichtige Informationen verloren gehen. Außerdem können sie als störend empfunden werden und zur Ablehnung des Programms führen. Zu wenige Informationen können sich allerdings negativ auf die Nachvollziehbarkeit auswirken. Fühlt sich der Anwender überfordert oder versteht nicht, wie das Programm handelt, kann dies ebenfalls zur Nichtakzeptanz führen.

Anders als Parallel Studio und Prism steht bei Patty die Automatisierung mehr im Vordergrund. Es verlässt sich in seiner Ausführung nicht auf Entwicklerentscheidungen, sondern führt diesen durch den Parallelisierungsvorgang und teilt seine Ergebnisse mit.

5 IMPLEMENTIERUNG

In diesem Kapitel wird die Implementierung der Benutzerschnittstelle von Patty beschrieben. Eingangs wird auf die grundsätzliche Umsetzung der Schnittstelle eingegangen und ihr Aufbau beschrieben. Im Anschluss daran wird die der Schnittstelle zugrunde liegende Implementierungslogik erläutert.

5.1 Aufgaben der Schnittstelle

Die entwickelte Schnittstelle soll die Steuerung des automatischen Parallelisierungswerkzeugs implementieren. Hierfür muss es die im Kapitel 4.3 definierten Anforderungen erfüllen, sowie die dort definierten Eingabeparameter entgegennehmen können.

5.1.1 Einbindung in Visual Studio

Als Resultat aus den in Kapitel 3 über die Verwandte Arbeiten gewonnen Erkenntnissen wird die Benutzerschnittstelle als *Plug-In* in das Microsoft Visual Studio integriert. Seit Visual Studio 2013 gelten *Add-Ins* als veraltet und wurden durch die nachfolgenden *Visual Studio Packages* (auch *VSPackage*) ersetzt. Ein Visual Studio Package stellt eine Klassenbibliothek dar, welche beispielsweise beim Starten von Visual Studio oder durch einen Menüeintrag geladen werden kann. Für das Erstellen eines *VSPackage* gibt es drei verschiedene Möglichkeiten:

1. *Menu Command*
2. *Tool Window*
3. *Custom Editor*

Mit *Menu Command* können Menübefehle erstellt werden, welche über ein Kontextmenü aufgerufen werden können, oder in der Menüleiste von Visual Studio aufgeführt sind.

Ein *Tool Window* ist ein Fenster wie beispielsweise der Projektmappenexplorer. Unter Anderem können *Tool Windows* an verschiedenen Stellen im Visual Studio fest verankert werden.

Mit einem *Custom Editor* kann Visual Studio um einen Editor für eine weitere Programmiersprache erweitert werden, um auch andere Sprachen als die standardmäßigen (C++, C#, VB) in dieser Entwicklungsumgebung (*engl. integrated development environment/ IDE*) entwickeln zu können.

Da für die Benutzung von Patty zum einen viele Benutzerinteraktionen notwendig sind und es zum anderen graphische Ausgaben gibt, wird die Integration in Visual Studio als *Tool Window* realisiert.

5.1.2 Grundlegender Ansatz

Patty wurde in C# für das Microsoft Visual Studio 2012 entwickelt. Die Benutzerschnittstelle wird nach dem *Model-View-Controller(MVC)* [ES13] Prinzip entwickelt.

5.2 Aufbau der grafischen Benutzeroberfläche

Die für Patty verwendeten *ToolWindow* lassen sich in Visual Studio über den Menüeintrag „Ansicht“ (*engl. view*) öffnen. Patty besteht aus zwei dieser Fenster: `PattyToolWindow` und `ProgressAndResults`. Diese Fenster werden aus sogenannte Benutzersteuereinheiten (*engl.*

user controls) zusammengesetzt. Die verwendeten *User Controls* sind in Kapitel 5.3.3 zusammengefasst. `PattyToolWindow` hat die Steuerung des Parallelisierungsprozesses sowie die Verarbeitung der Benutzereingaben zur Hauptaufgabe. Beispielsweise wird hier der Dateipfad zur zu parallelisierenden *Solution* angegeben oder es werden die Eingabeparameter definiert. `ProgressAndResults` übernimmt die Veranschaulichung des Vorgangs sowie die Ausgabe der Ergebnisse. Um die Nachvollziehbarkeit zu erleichtern, wird der aktuelle Fortschritt bildlich dargestellt. Außerdem lassen sich hier die im Konzept (Kapitel 4.3.3) definierten Ergebnisse von untersuchten Codestellen begutachten. Die Trennung von Steuerungsansicht und Ergebnisansicht dient der Verständlichkeit und soll die Gebrauchstauglichkeit erhöhen.

Der *Controller* wird durch den `VSGUIController` implementiert und dient als Ansprechstelle für alle Benutzersteuerelemente. Er ist als Einzelstück (*engl. Singleton*) implementiert. An ihn laufen die Anfragen aus den Benutzeransichten und werden entsprechend weiter verteilt.

Das Model, realisiert durch die Klasse `VSGUIModel`, verwaltet die Daten und steuert den gesamten Prozess. Zur Ausführung des Parallelisierungsprozesses wird mit der von .NET bereitgestellten `BackgroundWorker-Klasse` [BAC] gearbeitet, welche die ihnen zugeteilte Berechnungen auf einem separaten Faden ausführen.

5.2.1 Beobachtermuster des Models

Um die Benutzeransichten (*engl. view*) immer aktuell zu halten, wird das sogenannte Beobachter-Entwurfsmuster (*engl. observer pattern)* [ES13] verwendet. Hierfür erbt die Klasse `VSGUIModel` als Subjekt des Beobachtermusters von der Klasse `Observable` die typischen Methoden `Attach`, `Detach` und `Notify`, sowie eine Liste, in der die Beobachter gespeichert werden. Alle Beobachter des Models implementieren das Interface `ModelObserver`, welches sie mit einer `Update`-Methode versorgt.

Hat sich das Model verändert und möchte die Beobachter darüber benachrichtigen, so wird über die `Notify`-Methode bei allen im Model gespeicherten Beobachtern die `Update`-Methode ausgeführt. Als Parameter übergibt das Model seinen derzeitigen Zustand, woran auch die zu aktualisierenden Elemente bestimmt werden.

5.3 Komponenten der Implementierung

Nachfolgend werden die verschiedenen Komponenten, aus denen die Benutzerschnittstelle aufgebaut ist oder mit denen sie arbeitet, aufgeführt. Es werden die verwendeten Aufzählungen (*engl. enumeration*, kurz *enum*) genannt, sowie die im Werkzeug umgesetzten Operationsmöglichkeiten erklärt. Anschließend werden die implementierten Benutzersteuerelemente und deren Aufgabe veranschaulicht und das implementierte Zustandsmuster erläutert. Abschließend wird auf die Visualisierung der parallelisierbaren Quelltextsegmente sowie auf die der Abhängigkeitsgraphen eingegangen.

5.3.1 Aufzählungen

Um das Anzeigeverhalten der grafischen Benutzerschnittstelle (*engl. graphical user interface/GUI*) zu beschreiben, werden verschiedene *Enumerations* verwendet. Durch *Enumerations* kann bestimmt werden, was in den Benutzeransichten gerade angezeigt wird oder wie sie sich verhalten sollen. Weiter definieren sie Parameter, welche für die Programmausführung notwendig sind.
Die GUI verwendet folgende Aufzählungen:

- `GUIProcessStep`:
  Anhand dieser Aufzählung wird bestimmt, was aktuell in den *Tool Windows* angezeigt werden soll. Sie wird verwendet, um den im *Model* gespeicherten Zustand der GUI (Kapitel 5.3.4) zu verändern.
- `StatusImage`:
  Die hier aufgezählten Elemente repräsentieren jeweils ein Bild, welches in die Benutzeransicht geladen wird und den Prozessverlauf verdeutlichen soll.
- `OperationMode`:
  Bestimmt, wie Patty mit dem übergebenen Quelltext vorgehen soll. Setzt die in Kapitel 4.3.1 definierten Möglichkeiten um.
- `PatternDetection`:
  Parameter, der bestimmt, auf welchen Parallelisierungskandidaten [H13] der Code untersucht werden soll.
- `CollitionDetection`:
  Parameter, welcher festlegt, welche Parallelisierungsmöglichkeit bei einer Überschneidung bevorzugt werden soll.
- `ProcessingMode`:
  Parameter, mit welchem bestimmt wird, ob der Parallelisierungsvorgang Schritt für Schritt durchgeführt oder am Stück durchlaufen soll.

5.3.2 Operationsmodi

Wie im Kapitel 4.3 definiert ist eine Anforderung an Patty, dem Benutzer möglichst flexible Einstiegsmöglichkeiten zu bieten. In der Implementierung wurden deshalb folgende Operationsmodi umgesetzt:

- Quellcodeanalyse zur Identifizierung möglicher Parallelisierungsabschnitte
- Automatische Parallelisierung von annotiertem Code
- Komplettdurchlauf als Kombination der ersten beiden Modi

Quellcodeanalyse zur Identifizierung möglicher Parallelisierungsabschnitte

Zunächst werden eine statische und dynamische Analyse des Quellcodes durchgeführt, um Daten- und Kontrollflussabhängigkeiten zu erkennen. Der so analysierte Code wird nun mit einem Katalog von Parallelisierungsmustern verglichen und geeignete Zielmuster (Pipeline, Master/ Worker) für jeden identifizierten Codeabschnitt werden bestimmt. Wurde für einen Codeabschnitt ein geeignetes Muster gefunden, so werden die zu parallelisierenden Elemente im Code mit *TADL*-Kommentaren versehen

Automatische Parallelisierung von annotiertem Code

Wandelt *TADL*-annotierten Quelltext um, sodass dieser parallel ausführbar ist.

5.3.3 Benutzersteuerelemente

Die in die Entwicklungsumgebung eingebetteten Fenster von Patty werden aus verschiedenen Benutzersteuerelementen (*engl. user control*) zusammengesetzt. Benutzersteuerelemente kapseln eine Interaktionsmöglichkeit mit Patty und zeichnen sich so durch ihre Wiederverwendbarkeit aus. Sie befinden sich hauptsächlich im Namensraum (*engl. namespace*) `FRG.VS12Package.UserControls`. Lediglich die den Zuständen zugeordneten *User Controls* befinden sich im Namensraum `FRG.VS12Package.StateControls`.

FRG.VS12Package.UserControls

Bis auf die nachfolgend beschriebenen `StateControls` befinden sich alle weiteren Benutzersteuerelemente in diesem Namensraum. Die Benutzersteuerelemente geben ihre Benutzerinteraktionen an den `VSGUIController` zur Verarbeitung. Jedes Steuerelement hat seine eigene Aufgabe, wie zum Beispiel das Entgegennehmen von Eingabedaten oder das Festlegen von für die Durchführung wichtigen Parametern. Im Folgenden werden die Benutzersteuerelemente und ihre Aufgaben in alphabetischer Reihenfolge beschrieben.

ArchitecturePatternControl

Mit diesem Steuerelement kann der Benutzer festlegen, mit welchen Parallelisierungsmethoden (*Pipeline* oder *Master/Worker*) gearbeitet werden soll. Er kann sich entweder für eines der beiden Vorgehen entscheiden, oder den Quelltext nach beiden Mustern bearbeiten lassen. Dieses Steuerelement stellt somit die im Kapitel 4.2.1 geforderte Eingabemöglichkeit 3 zur Verfügung.

ExecutionControl

Mit dieser *User Control* wird der Parallelisierungsvorgang angestoßen oder abgebrochen beziehungsweise zurückgesetzt.

OperationModeControl

Stellt die in Kapitel 5.3.2 definierten Operationsmodi in einer *Combobox* zur Verfügung und bestimmt so, nach welchem Patty arbeiten soll. Setzt die Anforderung nach flexiblen Einstiegsmöglichkeiten (4.3.1) um.

ProcessingModeControl

Setzt die im Kapitel 4.2.1 definierte Eingabeoption 4 um, indem es dem Benutzer die Auswahl zwischen automatischem Volldurchlauf oder Schritt-für-Schritt-Durchführung bietet.

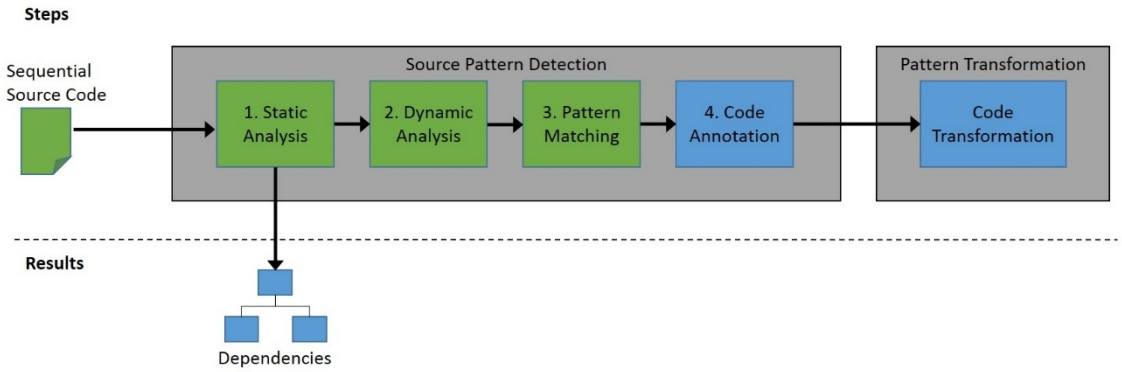

Abbildung 9: Visualisierung des Prozessverlaufs sowie der berechneten Ergebnisse

ProgressAndResultControl

Dieses Steuerelement kombiniert die Elemente, welche den Inhalt des `ProgressAndResults`-Fensters (Kapitel 5.1.2) realisieren. Dem Anwender sollen hier der Programmfortschritt und die Ergebnisse präsentiert werden. Es besteht dementsprechend aus zwei Teilen und erfüllt die in Kapitel 4.3 definierten Anforderungen A2 und A3.

Auf der linken Seite wird der Programmfortschritt anhand von Bildern gezeigt. In Abbildung 9 wird die Visualisierung des Programmfortschritts dargestellt. Bereits abgeschlossene Programmschritte werden grün markiert. Noch ausstehende Berechnungen sind blau hinterlegt. Die von

Patty errechneten und für den Benutzer einsehbaren Ergebnisse werden unterhalb der Fortschrittsansicht angezeigt.

Die rechte Seite bietet dem Anwender die Möglichkeit, von Patty errechnete Zwischenergebnisse auszugeben. Hierfür wird das Benutzersteuerelement `ResultControl` verwendet.

ProgramArgumentsControl

Für die dynamische Quelltextanalyse benötigt Patty Eingabeparameter. Die `ProgramArgumentControl` bietet dem Anwender die Möglichkeit, diese zu übergeben. Übergebene Parameter werden in der Klasse `Options` gespeichert und so für die Durchführung bereitgestellt.

ResultControl

Ermöglicht es dem Benutzer, von Patty während der Ausführung generierte Zwischenergebnisse anzuzeigen. Die einsehbaren Zwischenergebnisse entsprechen den in der im Kapitel 4.3 gestellten Anforderung A3. Um Ergebnisse anzuzeigen, wählt der Benutzer den Ausführungsschritt und die Klasse, für welche er sie einsehen möchte. Die Klasse wird dann in der *IDE* geöffnet. Je nach Prozessschritt stehen dann entsprechend der Anforderung A3 eine Färbung der Parallelisierungskandidaten (siehe Kapitel 5.3.5), die Abhängigkeitsgraphen (siehe Kapitel 5.3.6), der *TADL*-annotierte Quelltext oder der bereits umgewandelte Quellcode zur Verfügung.

SettingsControl

Neben den beiden Tool Windows existiert ein Fenster zum Vornehmen von Einstellungen. Der Inhalt dieses Fensters wird durch dieses Steuerelement definiert. Der Benutzer muss hier die Pfade zu den von Patty benötigten Bibliotheken angeben. Optional kann noch der Pfad zum Speicherplatz des zu untersuchenden Quelltexts angegeben werden. Dieser wird gespeichert und verwendet, um nicht den Pfad zur *Solution* immer wieder neu eingeben zu müssen. Alle eingegebenen Pfade werden in der Klasse `Options` gepeichert. Zusätzlich wird das Steuerelement `OperationModeControl` angeboten.

SolutionControl

Hier gibt der Benutzer den Pfad zur C#-Solution, also zu den zu untersuchenden Quelltexten ein.

TunableControl

Befindet sich Patty im Operationsmodus, welcher bereits *TADL*-annotierten Quellcode in parallel ausführbaren Code übersetzen soll, so wird zur Eingabe des Quelltextes dieses Steuerelement verwendet.

FRG.VS12Package.StateControls

Diese *StateControls* implementieren zusätzlich die Schnittstelle (*engl. Interface*) `StateControl`, welche sie mit den Methoden `Update` und `Reset` ausstattet. `Update` wird verwendet, um die `StateControl` an den derzeitigen Zustand (Kapitel 5.3.4) anzupassen. Dies bedeutet, hier wird festgelegt, welche Interaktionsmöglichkeiten dem Benutzer derzeit zur Verfügung stehen sollen. `Reset` setzt alle in diesem Benutzersteuerelement verwendeten Eingabefelder zurück.

Die *StateControls* sollen die Möglichkeit bieten, Daten und Eingabeparameter für die Operationsmodi (5.3.2) zu übergeben. Hierfür kombinieren sie die zuvor aufgezählten *UserControls*. Je nach Modus wird dann das entsprechende Steuerelement geladen. In Abbildung 10 ist das zur Steuerung des vollständigen Parallelisierungsvorgangs verwendete `StateControl` zu sehen. Patty besitzt die Folgenden *StateControls*:

SPDStateControl

SPD steht hier für *Source Pattern Detection*, also für Erkennung von parallelisierbaren Codeabschnitten. Das Steuerelement wird verwendet, wenn der Parallelisierungsprozess von der statischen Analyse bis hin zur *TADL*-Annotation des Quelltexts durchgeführt werden soll.

TPTStateControl

TPT steht hier für *Target Pattern Transformation*, also die Umwandlung von gefundenen Parallelisierungspotential in tatsächlich parallel ausführbaren *Code*. Das Benutzersteuerelement wird demnach verwendet, wenn bereits *TADL*-annotierter Quellcode vorliegt und dieser lediglich transformiert werden soll.

FullStateControl

Wenn der Parallelisierungsprozess vollständig abgearbeitet werden soll, so wird dieses Steuerelement verwendet. Es ist in Abbildung 10 exemplarisch dargestellt.

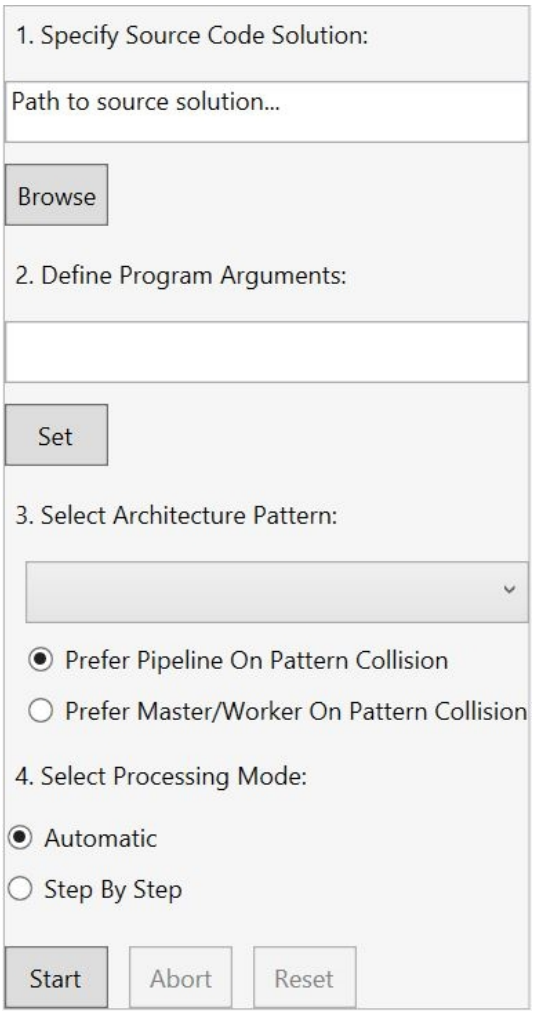

Abbildung 10: StateControl zur Steuerung des Parallelisierungsvorgangs

5.3.4 Zustände der Schnittstelle

Zur Beschreibung des Zustands der grafischen Benutzeroberfläche wird das Zustands-Entwurfsmuster *(engl. state pattern)* verwendet. Die Zustände repräsentieren die verschiedenen Operationsmodi (siehe 5.3.2) von Patty und verwalten die `GUIInputSteps` (5.3.1).
In den Zuständen wird gespeichert, was derzeit dem Benutzer in den *Tool Windows* angezeigt werden soll. Sie werden im Namensraum `FRG.VS12Package.States` angesiedelt. Der derzeitige Zustand wird im `VSGUIModel` bereitgestellt und verwaltet. Die Klasse `GUIState` stellt den abstrakten Zustand dar. Das in die Unterklassen ausgelagerte Verhalten ist die Anpassung der Benutzeransichten an eine Änderung des Prozessfortschritts (siehe `InputSteps` im Kapitel 5.3.1).
In den Zuständen werden ebenfalls die entsprechenden Benutzersteuerelemente der *Tool Windows* sowohl gespeichert als auch verwaltet. Diese Steuerelemente finden sich im Namensraum `FRG.VS12Package.StateControls`. Eine Beschreibung der Funktion und des Aufbaus dieser Elemente findet sich im Kapitel 5.3.3. Ändert sich der Zustand, so wird auch gleich das zugehörige Benutzersteuerelement angepasst und muss in der durch den *Model-Observer* (siehe Kapitel 5.2.1) ausgelösten Aktualisierung der Benutzeransichten nur noch geladen werden.

Änderung des Zustandes

Wählt der Benutzer einen neuen Operationsmodus (siehe 5.3.2), so ändert sich der im Model gespeicherte Zustand entsprechend und passt so die *Tool Windows* an. Aber auch die Zustände erfahren im Verlauf der Programmausführung durch Benutzereingaben und Programmabläufe Veränderungen. Diese Änderungen werden durch die `GUIProcessStep` (siehe Kapitel 5.3.1) repräsentiert. Übergibt man einem Zustand einen neuen `GUIProcessStep`, so wird in diesem bestimmt, welche Benutzersteuerelemente (siehe Kapitel 5.3.3) zur Verfügung stehen sollen. Anschließend wird direkt das dem Zustand zugehörige Steuerelement aktualisiert.

Die Zustände repräsentieren also die Operationsmodi, die `GUIProcessStep`s definieren, welche Interaktionsmöglichkeiten dem Benutzer im derzeitigen Modus zur Verfügung stehen.

5.3.5 Visualisierung der Parallelisierungskandidaten

Um die Nachvollziehbarkeit zu verbessern, sollen wie in Kapitel 4.3.2 festgelegt, Codestellen, welche Parallelisierungspotential haben, hervorgehoben werden. Zur Visualisierung der Parallelisierungskandidaten [H13] wurde die Visual Studio Erweiterung *Editor Text Adornment* verwendet, welche den im Visual Studio Editor angezeigten Quellcode, beispielsweise durch eine Einfärbung von Codezeilen, ausschmückt.

Diese Erweiterung wurde in Form des `CandidateVisualizer` realisiert. Ein *Editor Text Adornment* besteht zunächst aus zwei Komponenten, einer Fabrik und der Klasse, welche die Visualisierung implementiert (im Folgenden „Visualisierer“ genannt). Für die Parallelisierungskandidatenvisualisierung in Patty wurde zusätzlich die Klasse `CandidateDepository` eingeführt, welche die von Patty untersuchten Klassen den während des Parallelisierungsprozesses gefundenen Kandidaten zuordnet.

Fabrik und Vorverarbeitung

Die Methode `TextViewCreated` der Fabrik wird aufgerufen, sobald ein neuer Quelltext in Visual Studio erscheint. Ihre Aufgabe ist die Vorverarbeitung und anschließende Initialisierung der Visualisierer. Hierfür lädt sie zunächst dem aktuellen Quelltext zugehörigen Parallelisierungskandidaten. Weiter lädt sie den vollständigen Quelltext und spaltet ihn zeilenweise auf, um den Visualisierer zu initialisieren (Erläuterung der Initialisierung im folgenden Absatz). Diese Aufspaltung ist nötig, da der Visualisierer den im Editor dargestellten Code ebenfalls zeilenweise einliest.

Wurde für das aktuelle Dokument bereits Parallelisierungskandidaten ermittelt, so wird der Visualisierer erstellt.

Visualisierer

Hauptaufgabe der Klasse `CandidateVisualizer` ist das Hervorheben von Parallelisierungskandidaten. Kandidaten, die von der Analyse noch nicht ausgeschlossen wurden, werden grün hinterlegt. Diejenigen Kandidaten, die bereits verworfen wurden, werden rot hinterlegt.

Um dies umzusetzen, wurde Patty um die Klasse `CandidateSpan` erweitert, deren Aufgabe es ist, zu jedem Kandidaten den Anfangs- und Endpunkt im Quelltext zu speichern. Zudem wurde die Klasse `ParallelizationCandidate` um mehrere Attribute und Methoden erweitert. Diese verwalten den Pfad zu den gespeicherten .dgml-Dateien (siehe 5.3.6), den Pfad zur zugehörigen Klassendatei, sowie den dem Kandidaten zugehörigen Quelltext und bestimmen, ob der Kandidat noch gültig ist.

Initialisierung

Zur Einfärbung des Quelltexts können nicht die in den Parallelisierungskandidaten gespeicherten Start- und Endpunkte verwendet werden, da diese ermittelt werden, bevor der Quelltext, beispielsweise durch die `Logger`-Anweisungen [H13], ergänzt wird. Um diese Veränderungen miteinzubeziehen, müssen die Start und Endpunkte der Färbung im Editor in der Initialisierung zunächst neu berechnet werden.

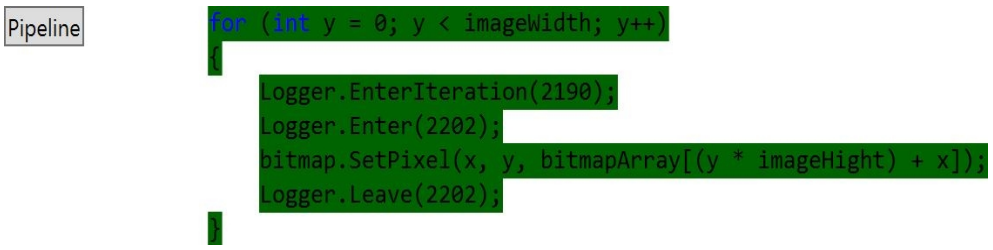

Abbildung 11: Codeausschnitt mit Einfärbung einer parallelisierbaren `for`-Schleife

Wird also ein neuer `CandidateVisualizer` erstellt, so initialisiert er zunächst die `CandidateSpan` für die zugehörigen Kandidaten. Hierfür durchläuft er für jeden Kandidaten nacheinander alle Codezeilen im Editor von oben nach unten und sucht den Beginn des Kandidatenquelltexts. Hat er diesen gefunden, so speichert er die Codestelle im zugehörigen `CandidateSpan` als Startpunkt. Nach Auffinden des Startpunktes werden kommende Codezeilen mit denen des Kandidaten verglichen und bei Übereinstimmung abgehakt. Für die Suche arbeitet der Visualisierer also wie eine Art Kellerspeicher (*engl. stack*), welcher immer dann ein Element entfernt, wenn eine passende Quelltextzeile gefunden wurde. Editorzeilen, welche nicht zum derzeit gesuchten Eintrag passen (wie zum Beispiel die `Logger`-Instrumentierungen), werden also übersprungen und die nächste Zeile wird überprüft. Erreicht dieser Prozess das Ende des Kandidaten, wird dieses ebenfalls notiert und der `CandidateSpan` wird gespeichert.

Färbung

Die im Visualisierer enthaltene Methode `OnLayoutChanged` wird immer dann aufgerufen, wenn im Editor Zeilen auf irgendeine Art verändert werden (z.B. Verschwinden oder Erscheinen beim Scrollen). Die veränderten Codezeilen werden nacheinander an die `CreateVisuals`-Methode übergeben.

Liegt die betrachtete Zeile innerhalb eines der zuvor ermittelten `CandidateSpan`, so wird sie entsprechend Kapitel 4.3.2 grün oder rot gefärbt. Eine solche Einfärbung ist in Abbildung 11 zu sehen. Ist die aktuelle Zeile gleichzeitig der Beginn eines Parallelisierungskandidaten, so wird sie mit einem Knopf (*engl. button*) versehen. Dieser Knopf trägt als Beschriftung die Art des Kandidaten (*Taskgraph* oder *Pipeline*), um dem Benutzer klar zu machen, welche Parallelisierungsstrategie an dieser Stelle anwendbar sein könnte. Betätigt man den Knopf, so öffnet sich der zugehörige Abhängigkeitsgraph.

5.3.6 Visualisierung der Abhängigkeitsgraphen

Zur Darstellung der Abhängigkeitsgraphen wird die *XML*-basierte *Directed Graph Markup Language (DGML)* [DGML] verwendet, da sich die so erstellten Graphen direkt in Visual Studio öffnen und bearbeiten lassen. Um die .dgml-Dateien zu erstellen, wurde Patty die Klasse `DGMLWriter` hinzugefügt. Diese wird mit einem `XmlTextWriter` [XML] initialisiert, welcher dann verwendet wird, um die Dateien zu erstellen. Der `DGMLWriter` erstellt zu einem übergebenen `NodeDependenceGraph` (siehe [H13]) die zugehörige Datei entsprechend den in Kapitel 4.3 definierten Anforderungen. Zu Beginn werden die Einträge für die Knoten,

anschließend die der Kanten, gefolgt von der Definition der Kategorien, erstellt. Abbildung 12 zeigt den Aufbau einer solchen .dgml-Datei.

Knoten

Um die Knoten später mit den Kanten in Verbindung bringen zu können, müssen sie mit einer eindeutigen ID versehen werden. Hierfür wird die ohnehin schon vorhandene `SyntaxNodeId` [H13] verwendet. Beschriftet werden die Knoten entsprechend dem Konzept (Kapitel 4.3.2) mit dem zugehörigen Quelltext.

Kanten

Die Kanten verwenden die in den Knoten gespeicherten IDs, um Start- und Zielknoten zu bestimmen. Für die Kanten sind die sieben in Patty möglichen Relationen (siehe [H13]) definiert. Realisiert werden die Kantenrelationen, indem für jede Relation eine Kategorie definiert wird. Hat die Kante die Relation *None*, also keine Abhängigkeit, so wird dem Graphen keine Kante hinzugefügt.

```
<?xml version="1.0" encoding="utf-8"?>
<DirectedGraph xmlns="http://schemas.microsoft.com/vs/2009/dgml">
  <Nodes>
    <Node Id="2196" Label="y &lt; imageWidth" />
    <Node Id="2202" Label="bitmap.SetPixel(x, y, bitmapArray[(y * imageHight) + x]);" />
    <Node Id="2199" Label="y++" />
  </Nodes>
  <Links>
    <Link Source="2199" Target="2196" Category="Write After Read" />
    <Link Source="2199" Target="2202" Category="Write After Read" />
  </Links>
  <Categories>
    <Category Id="RAW" Label="RAW" Stroke="Red" />
    <Category Id="WAR" Label="WAR" Stroke="Blue" />
    <Category Id="WAW" Label="WAW" Stroke="Orange" />
    <Category Id="LopCarried" Label="LopCarried" Stroke="Black" />
    <Category Id="None" Label="None" Stroke="Black" />
    <Category Id="ControlFlow" Label="ControlFlow" Stroke="Grey" />
    <Category Id="Transitiv" Label="Transitiv" Stroke="Black" />
    <Category Id="Several" Label="Several" Stroke="Black" />
  </Categories>
</DirectedGraph>
```

Abbildung 12: Aufbau einer .dgml-Datei. Zunächst werden die Knoten, dann die Kanten und abschließend die Kategorien definiert.

Kategorien

Die *DGML* bietet eine Möglichkeit, Knoten und Kanten in sogenannten Kategorien (*engl. Category*) zusammenzufassen. Für jede der Kantenrelationen wird eine eigene Kategorie angelegt, welcher die Kanten entsprechend zugeordnet werden. Neben der textuellen Information wird in der Kategorie ebenfalls ein Farbschema festgelegt, um die Kanten im Abhängigkeitsgraphen besser darstellen zu können. Die in Patty vorkommenden Kantentypen wurden in [H13] definiert.

6 EVALUIERUNG

Aufgabe von Patty ist das Umwandeln von seriellen Quelltexten, sodass diese parallel ausführbar sind. Im Zuge dieser Arbeit werden zum einen die entwickelte Benutzerschnittstelle hinsichtlich ihrer Gebrauchstauglichkeit und zum anderen der Geschwindigkeitsvorteil von Patty gegenüber einem anderen Werkzeug und einer manuellen Parallelisierung evaluiert. Die Evaluierung erfolgt in Form einer Entwicklerstudie mit Studenten und wissenschaftlichen Mitarbeitern am KIT. Dieses Kapitel beschäftigt sich zunächst mit dem Aufbau sowie dem Ablauf der Studie. Anschließend werden die Ergebnisse der Entwicklerstudie aufgelistet und ausgewertet.

6.1 Versuchsaufbau

Die Probanden werden in drei Gruppen aufgeteilt, wobei jede Gruppe nach einer anderen der folgenden Methoden arbeitet. Die Probanden arbeiten alleine an jeweils separaten Rechnern.

Herangehensweisen:

- Manuelle Untersuchung auf Parallelisierungspotential ohne Hilfsmittel
- Parallelisierung mit dem Intel Parallel Studio
- Parallelisierung mit Patty

Aufgabe der Probanden ist das Ermitteln von Parallelisierungspotential bei einem gegebenen seriellen Quelltext.

Zur späteren Auswertung wird während der Studie ein Bildschirmvideo mitgeschnitten. Weiter erhalten die Probanden jeweils einen Fragebogen, welcher die subjektive Einschätzung der Versuchspersonen ermittelt. Die Fragebögen gibt es in zwei Versionen; einen für die manuelle und einen für die werkzeugunterstütze Gruppe.

Die Ergebnisse der Testpersonen werden anhand folgender Aspekte beurteilt.

Vergleichsaspekte:

- Anzahl identifizierter Codestellen
- Auswertung der Bildschirmvideos
- Auswertung der Fragebögen

Um möglichst vergleichbare Probandengruppen zu bilden, werden die Probanden vorab nach ihrer Erfahrung im Bereich Softwareentwicklung und speziell Multicore, sowie nach einer Selbsteinschätzung ihrer Erfahrung auf dem Gebiet der parallelen Entwicklung auf einer Skala von 1 (sehr gut) bis 6 (ungenügend) befragt. Anhand dieser Einschätzung werden die Gruppen so gebildet, dass die Mittelwerte über die Einschätzungen möglichst gleich sind und allen Gruppen sowohl erfahrene als auch unerfahrene Teilnehmer zugeordnet sind.

6.1.1 Aufgabenstellung

Die Probanden erhalten einen seriellen Quelltext mit der Aufgabe, diesen auf Parallelisierungspotential zu untersuchen. Sie sollen also die Quelltextstellen identifizieren, welche sich zur Parallelisierung eignen. Konkret lautet die Aufgabenstellung

> Finden sie alle Quellcodestellen, welche sich zur nebenläufigen Ausführung eignen.

Der Quelltest steht den Probanden dabei sowohl elektronisch, als auch ausgedruckt bereit.
Für die Aufgabe haben sie eine Stunde reine Bearbeitungszeit.

Nicht Teil der Aufgabe ist das Umwandeln in tatsächlich parallel ausführbaren Quellcode.

6.1.2 Herangehensweise

Die Versuchsteilnehmer arbeiten jeweils nach der ihrer Gruppe zugeordneten Methode. Je nach Testgruppe werden den Probanden gruppenspezifisch die nachfolgenden Hilfsmittel zur Seite gestellt.

Manuelle Untersuchung auf parallelisierungspotential ohne Werkzeug

Die Testgruppe, welche manuell vorgeht, darf den in Visual Studio bereits integrierten Leistungs-Assistenten verwenden. Hiermit kann ermittelt werden, an welchen Stellen die *CPU* viel Rechenaufwand betreibt.

Parallelisierung mit dem Intel Parallel Studio

Die Probanden dieser Versuchsgruppe bekommen das Intel Parallel Studio (Kapitel 3.4), eingebunden in Visual Studio, als Werkzeug zur Verfügung gestellt.

Parallelisierung mit Patty

Versuchspersonen in dieser Gruppe arbeiten mit Patty als Parallelisierungswerkzeug, was ebenfalls in Visual Studio eingebunden ist.

6.1.3 Vergleichsaspekte

Um die Ergebnisse der Versuchsgruppen vergleichen zu können, müssen dafür Kriterien definiert werden. Der Vergleich der Testgruppen wird anhand der folgenden Aspekte durchgeführt.

Auswertung der Bildschirmvideos

Die mitgeschnittenen Bildschirmvideos bieten die Möglichkeit, das Arbeiten der Testteilnehmer zu beurteilen. Anhand der Videos wird ermittelt, wie schnell die Teilnehmer mit der Aufgabe beginnen oder ob sie eventuell das ihnen bereitgestellte Werkzeug falsch verwenden. Gemessen werden außerdem die Zeitintervalle der gesamten Bearbeitungsdauer oder die Dauer vom Beginn der Testphase bis hin zur Entdeckung der ersten oder letzten Codestellen, welche Parallelisierungspotential haben.

Anzahl identifizierter Codestellen

Die Testteilnehmer halten das Ergebnis, zu welchem sie während der Testphase gekommen sind, fest, indem sie alle von ihnen als potentiell parallelisierbar ermittelten Quelltextstellen notieren. Nach Beendigung der Aufgabe werden die Ergebnisse auf die Anzahl der korrekt entdeckten Codestellen untersucht.

Auswertung der Fragebögen

Allen Probanden wird ein Fragebogen ausgeteilt. Diese Fragebögen bestehen aus einem Teil, welcher für alle Testpersonen gleich ist, sowie aus einem gruppenspezifischen Teil. Zunächst ermitteln die Fragebögen die Erfahrungen und Fähigkeiten der Teilnehmer. Die darauffolgenden Fragen sind für die manuelle und werkzeugunterstützen Teilnehmer verschieden. Die Gruppe der manuell arbeitenden Probanden wird nach Funktionalitäten befragt, welche ihnen bei der Parallelisierung nützen würden, wenn sie ein Werkzeug zur Seite gestellt bekämen. Bei der werkzeugunterstützen Gruppen wird das Werkzeug auf die Nutzererfahrung (engl. *user experience/UX*) geprüft. Die Fragen zur *User Experience* , welche in Abbildung 13 aufgelistet sind, orientieren sich am *UEQ* (*engl. user experience questionnaire*, deutsch: Fragebogen zum Benutzererlebnis) [UEQ]. Der *UEQ* ist ein standardisierter Fragebogen, welcher die *User Experience* der Interaktion mit einem Produkt misst, indem der Benutzer seine Erfahrung anhand von bipolaren Aussagen („einfach“ /“schwer“) auf einer siebenstufigen Skala angibt.

	1	2	3	4	5	6	7		
unverständlich	○	○	○	○	○	○	○	verständlich	1
leicht zu lernen	○	○	○	○	○	○	○	schwer zu lernen	2
wertvoll	○	○	○	○	○	○	○	minderwertig	3
langweilig	○	○	○	○	○	○	○	spannend	4
uninteressant	○	○	○	○	○	○	○	interessant	5
unberechenbar	○	○	○	○	○	○	○	voraussagbar	6
schnell	○	○	○	○	○	○	○	langsam	7
behindernd	○	○	○	○	○	○	○	unterstützend	8
kompliziert	○	○	○	○	○	○	○	einfach	9
sicher	○	○	○	○	○	○	○	unsicher	10
aktivierend	○	○	○	○	○	○	○	einschläfernd	11
erwartungskonform	○	○	○	○	○	○	○	nicht erwartungskonform	12
ineffizient	○	○	○	○	○	○	○	effizient	13
übersichtlich	○	○	○	○	○	○	○	verwirrend	14
unpragmatisch	○	○	○	○	○	○	○	pragmatisch	15
aufgeräumt	○	○	○	○	○	○	○	überladen	16

Abbildung 13: Die aus dem UEQ entnommenen Wertungsskalen

6.1.4 Versuchsquelltext

`RayTracer` ist eine C#-Anwendung zur Berechnung von Strahlenverläufen. Sie berechnet die Darstellung von 3D-Objekten in einem Raum aus Sicht eines Beobachters, indem die Schnitte der vom Beobachter ausgehenden Strahlen mit den Objekten berechnet werden. Der `RayTracer` besteht aus 13 Klassen und 173 Quelltextzeilen

6.2 Versuchsablauf

Der Versuch läuft nach einer kurzen Einführung in zwei Phasen ab; der Einführungs- und der Testphase.

6.2.1 Einführungsphase

Zunächst werden den Probanden die Aufgabenstellung erklärt, sowie die Fragebögen ausgeteilt. Anschließend erhalten die Testteilnehmer 15 Minuten Zeit, um sich mit den Quelltext vertraut zu machen. Diese Phase dient nur dem Kennenlernen des Codes. Es dürfen noch keine Parallelisierungswerkzeuge verwendet werden. Hilfsmittel, die nicht explizit der Parallelisierung dienen, wie beispielsweise der *Debug*-Modus, sind gestattet.

6.2.2 Testphase

Nach der Einführungsphase erfolgt die eigentliche Testphase mit der Untersuchung des Quellcodes. Diese hat eine maximale Dauer von einer Stunde. Spätestens nach Ablauf dieser Zeit werden die Fragebögen der Testpersonen eingesammelt. Mit der Abgabe des Fragebogens ist der Test beendet.

6.3 Versuchsauswertung

Die Entwicklerstudie wurde mit zehn Teilnehmern durchgeführt. Je drei Probanden arbeiteten mit Patty und manuell, vier Probanden arbeiteten mit Parallel Studio. Für die Auswertung wird zwischen erfahrenen und unerfahrenen Entwicklern unterschieden. Als erfahren sollen Entwickler mit mehr als sechs Jahren Erfahrung in der Softwareentwicklung und mindestens zwei Jahren im Bereich Multicore eingestuft werden. Alle weiteren Testpersonen werden in dieser Studie als unerfahren bezeichnet.

Zur Angabe der Ergebnisse wird ein Mittelwert von beispielsweise 2,2 mit einer Stichproben-Standardabweichung (engl. *standard deviation*) von 0,4 wie folgt angegeben: 2,2 [SD 0,4]. Die Ergebnisse werden auf die zweite Nachkommastelle gerundet.

6.3.1 Anzahl identifizierter Codestellen

Testteilnehmer, welche mit Parallel Studio gearbeitet haben, fanden im Schnitt 2,25 parallelisierbare Stellen, während in der Testgruppe Patty durchschnittlich drei Stellen identifiziert wurden. Alle gefundenen Quelltextabschnitte dieser beiden Gruppen sind korrekt. Nicht korrekte Stellen wurden nur von der manuell arbeitenden Gruppe notiert. Hätten diese Testpersonen die Abhängigkeiten innerhalb des fälschlich angegebenen Codesegments gekannt, so wäre dieser Fehler nicht passiert. Das Arbeiten mit Werkzeugen, die Datenabhängigkeiten bereitstellen, minimiert also das fälschliche identifizieren von Quelltextabschnitten. Abzüglich dieser falschen Stellen wurden im Schnitt zwei mögliche Codeabschnitte von der manuellen Gruppe identifiziert.

6.3.2 Auswertung der Fragebögen

Ausgewertet werden die subjektiven Angaben der Testteilnehmer auf ihren Fragebögen. Dargestellt in den nachfolgenden Diagrammen sind der Stichproben-Mittelwert des jeweiligen Aspekts sowie die Standardabweichung. Die Ausrichtung der gegensätzlichen Begriffe auf den Fragebögen ist zufällig gewählt. Die dortige Wertungsskala von 1 bis 7 wird auf einen Bereich von -3 bis +3 transformiert, wobei +3 die positivste und -3 die negativste Bewertung darstellt.

Aspekte Gebrauchstauglichkeit

Die hier untersuchten Aspekte sind Teil des UEQ und lassen sich in vier Kategorien einteilen (Definition aus [UEQ][1]):

[1] Genauer: http://www.ueq-online.org/index.php/what-is-the-user-experience-questionnaire/?lang=de, Verlässlichkeit entspricht hier Steuerbarkeit

- **Durchschaubarkeit**: Ist die Nutzung des Produkts einfach zu verstehen? Ist es einfach die Benutzung des Produkts zu erlernen?
- **Stimulation**: Ist es interessant und anregend das Produkt zu nutzen? Fühlt sich der Benutzer durch das Produkt motiviert?
- **Steuerbarkeit**: Hat der Benutzer den Eindruck, die Interaktion zu kontrollieren? Ist die Interaktion mit dem Produkt sicher und vorhersagbar?
- **Effizienz**: Kann der Benutzer mit dem Produkt schnell und effizient arbeiten? Ist die Benutzeroberfläche übersichtlich?

Die Aspekte sind in Vierergruppen den obigen Kategorien zugeteilt. Die ersten vier werden unter der Kategorie Durchschaubarkeit zusammengefasst, gefolgt von Stimulation, Steuerbarkeit und Effizienz. Als letzter Punkt wurde abgefragt, wie sehr sich der Benutzer durch das Werkzeug bei der Parallelisierung unterstützt fühlte. In den folgenden Tabellen, welche die Ergebnisse der spezifischen Fragenteils der werkzeugunterstützen Gruppen angeben, werden der Übersicht wegen nur die Mittelwerte angeben. Eine vollständige Darstellung inklusive der Standard-abweichungen findet sich im Anhang C. Die Beschriftungen der x-Achse sind die jeweils abgefragten Aspekte. Der letzte Punkt ist immer ein Gesamtdurchschnittswert für die jeweilige Kategorie. Die y-Achse gibt die Bewertung der Probanden an.

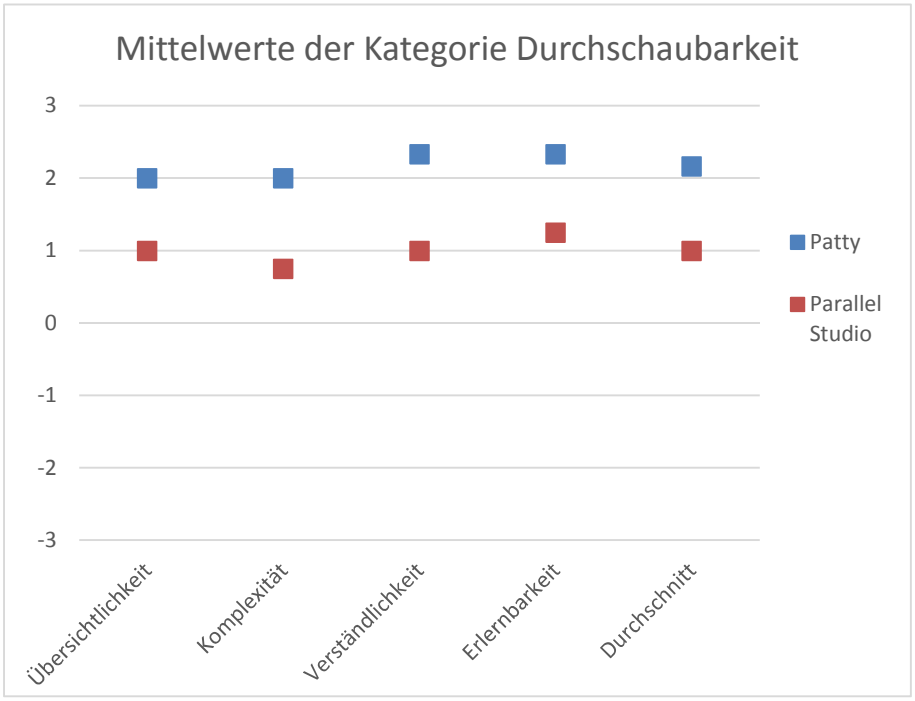

Abbildung 14: Ergebnisse der Kategorie Durchschaubarkeit. Darstellung nur der Mittelwerte

In der Kategorie „Durchschaubarkeit“ schneiden durchschnittlich sowohl Patty (2,17 [SD 0,68]) als auch Parallel Studio (1, [SD 1,75]) positiv ab. Abbildung 14 zeigt die Ergebnisse der Auswertung der einzelnen Aspekte sowie den Durchschnittswert über alle Aspekte. In dieser Kategorien wird Patty deutlich besser bewertet. Der Mittelwert ist um 1,17 höher, zudem ist die Streuung bei Patty geringer. Es wurde also von den Benutzern als nachvollziehbar empfunden. Das erklärte Ziel der Nachvollziehbarkeit wurde somit erreicht.

Die Ergebnisse für Patty und Parallel Studio der Kategorie „Stimulation“ sind in Abbildung 15 aufgezeigt. Betrachtet man die Aussagen der Testpersonen, so lässt sich zwischen den Ergebnissen von Parallel Studio (1,13 [SD 1,03]) und Patty (0,75 [SD 1,42]) kaum ein Unterschied feststellen.

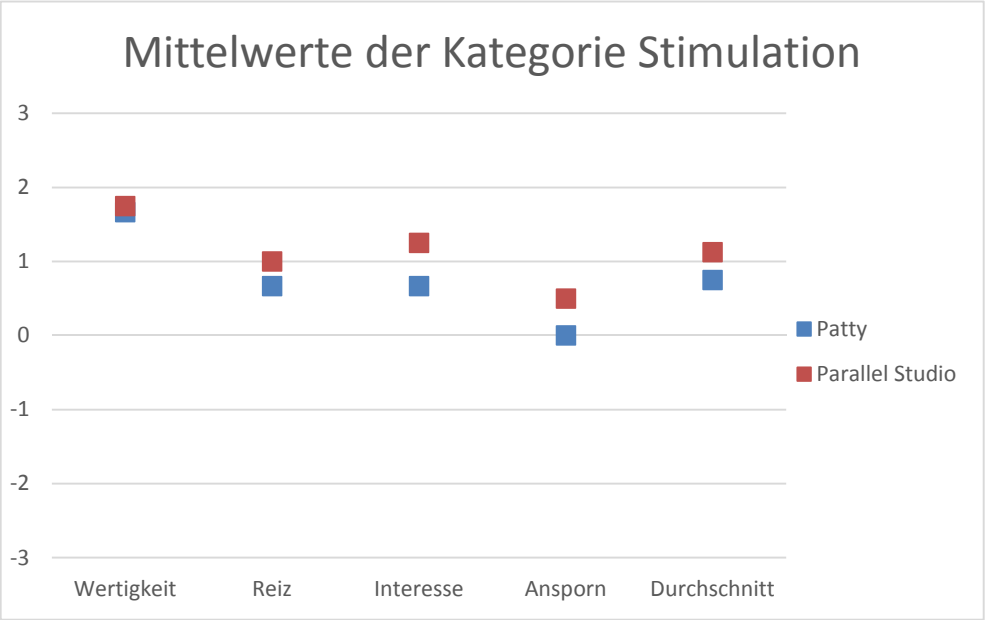

Abbildung 15: Erzielte Ergebnisse in der Kategorie "Stimulation". Dargestellt sind nur die Mittelwerte sowie ein Gesamtdurchschnittswert

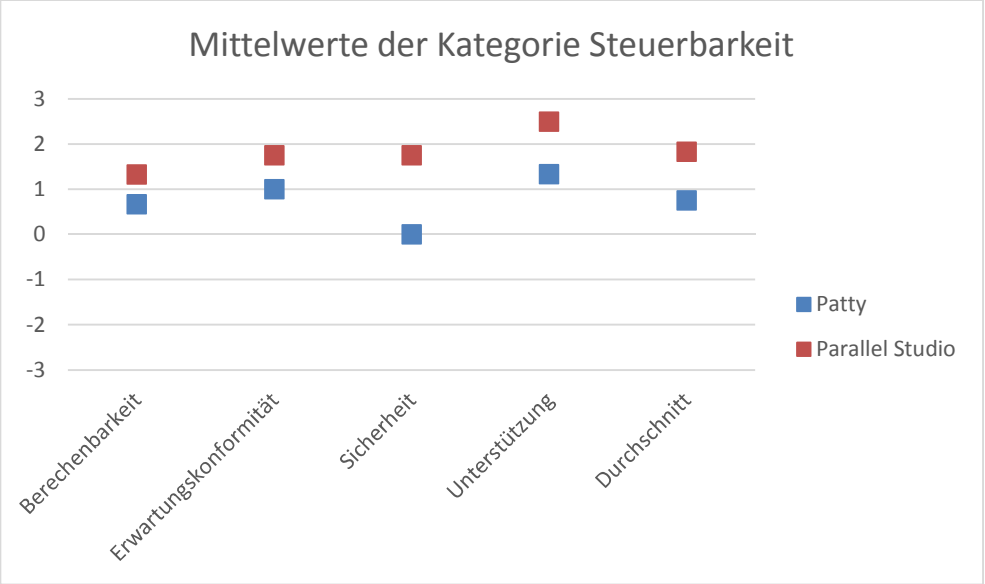

Abbildung 16: Erzielte Ergebnisse in der Kategorie "Steuerbarkeit". Dargestellt sind nur die Mittelwerte sowie ein Gesamtdurchschnittswert

Bei der Steuerung der Werkzeuge konnte insgesamt Parallel Studio (1,83 [SD 1,03]) ein klar besseres Ergebnis erzielen als Patty (0,75 [SD 0,83]). Die Ergebnisse werden in Abbildung 16 dargestellt. Patty schneidet hier in allen Punkten schlechter ab. Eine Erklärung hierfür ist, dass

Patty recht geradlinig durch den Parallelisierungsprozess führt und dem Benutzer hinterher seine Ergebnisse präsentiert. Beim Arbeiten mit Parallel Studio muss der Anwender deutlich aktiver werden. Teilweise stützt sich Parallel Studio auch ausschließlich auf Entscheidungen des Entwicklers (siehe 3.4). Welche Konsequenzen das haben kann, wird in der Ausewrtung der Bildschirmvideos erläutert.

Was die Effizienz anbelangt, so sind die Ergebnisse der beiden Werkzeuge wieder sehr ähnlich (Patty 1,33 [SD 0,58], Parallel Studio 1,31 [SD 1,59]) und tatsächlich wird sich bei der Videoauswertung zeigen, dass werkzeuggestütztes Arbeiten effizienter ist.

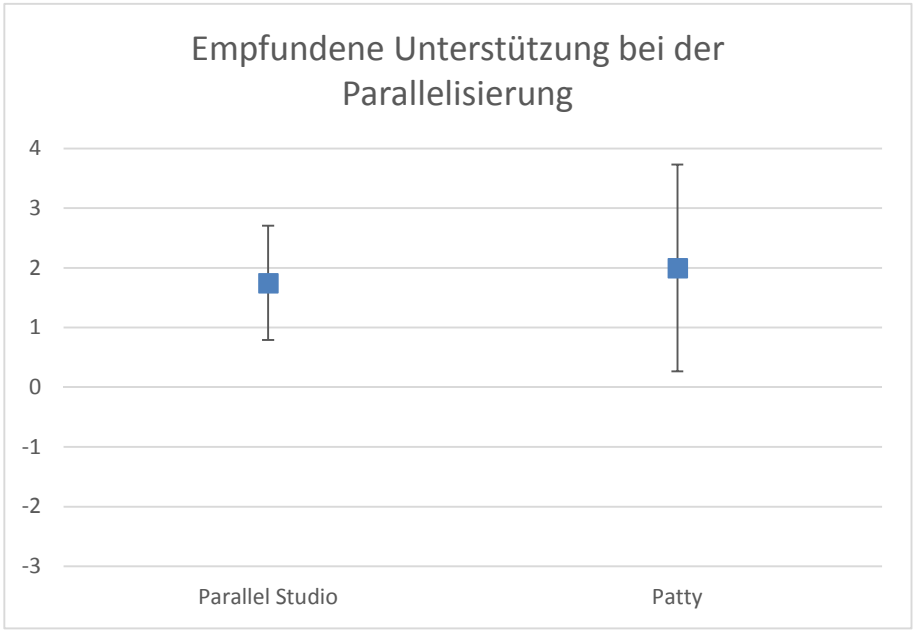

Abbildung 17: Auswertung, wie sehr sich die Probanden durch das jeweilige Werkzeug bei der Parallelisierung unterstützt fühlten

Ein weiterer Punkt der Befragung überprüfte, wie sehr sich die Testpersonen durch die Werkzeuge bei der Parallelisierung unterstützt fühlten. Die Ergebnisse sind in Abbildung 17 zu sehen. Insgesamt werden beide Werkzeuge bei der Parallelisierung als hilfreich empfunden (Patty 2 [SD 1,73], Parallel Studio 1,75 [SD 0,96]). Die große Streuung bei Patty kommt zustande, da Patty noch nicht zuverlässig alle Abhängigkeiten während der dynamischen Analyse erkennt (siehe [H13]). Eine dieser fehlenden Abhängigkeiten wurde von einem erfahrenen Entwickler entdeckt und entsprechend negativ fällt die Bewertung aus. Insbesondere wurde bemängelt, dass es nicht möglich ist, dem Programm diese gefundenen Abhängigkeiten beizubringen.

Gesamtnote

Die Gesamtnote wurde nicht auf einer Skala von 1 bis 7, sondern nach dem Schulnotenprinzip, also auf einer Skala von 1 (sehr gut) bis 6 (ungenügend), ermittelt. Auch hier konnte Patty mit einer 1,75 [SD 0,35] ein besseres Ergebnis als Parallel Studio mit 2,6 [SD 0,62] erzielen.

Zufriedenheit mit dem Ergebniss

Bemerkenswert ist, dass die Teilnehmer der manuellen Gruppe am zufriedensten (1,0 [SD 1,0]) waren, obwohl diese Gruppe die schlechtesten Ergebnisse erzielte. Teilnehmer dieser Gruppe haben die wenigsten parallelisierbaren Quelltextabschnitte entdeckt, außerdem wurden nur in dieser Gruppe nicht parallel ausführbare Codesegmente fälschlich als parallelisierbar angegeben.

In der Zufriedenheit wird die manuelle Gruppe unmittelbar gefolgt von Patty (0,67 [SD 0,58]). Testpersonen, die mit Parallel Studio gearbeitet haben, waren hingegen unzufrieden (-0,25 [SD 2,75]). Lediglich ein erfahrener Entwickler war mit seinem Ergebnis sehr zufrieden.

Gewünschte Werkzeugeigenschaften

Die manuell arbeitende Testgruppe wurde in ihrem Fragebogen nach Eigenschaften gefragt, welche sie sich von einem Werkzeug wünschen würden. Die Ergebnisse dieser Abfrage sind in Abbildung 18 visualisiert. Als am wichtigsten werden das Hervorheben von Codestellen mit Parallelisierungspotential (2,67 [SD 0,58]), sowie die Angabe von Datenabhängigkeiten (2,67 [SD 0,58]) genannt. Auch programmseitige Vorschläge einer möglichen Parallelisierungsstrategie (1,67 [SD 0,58]), die Darstellung des Kontrollflusses (1,67 [SD 1,15]) und die Angabe von Laufzeiten (2 [SD 1,73]) werden als hilfreich empfunden. Da Aufgabe der Studie das Auffinden von parallelisierbaren Quelltextstellen war, sind Aspekte wie Validierung (0,67 [SD 1,53]), Performanz-Optimierung (0,67 [SD 2,52]) und Quellcodemodellierung (-0,5 [SD 0,71]) eher unwichtig.

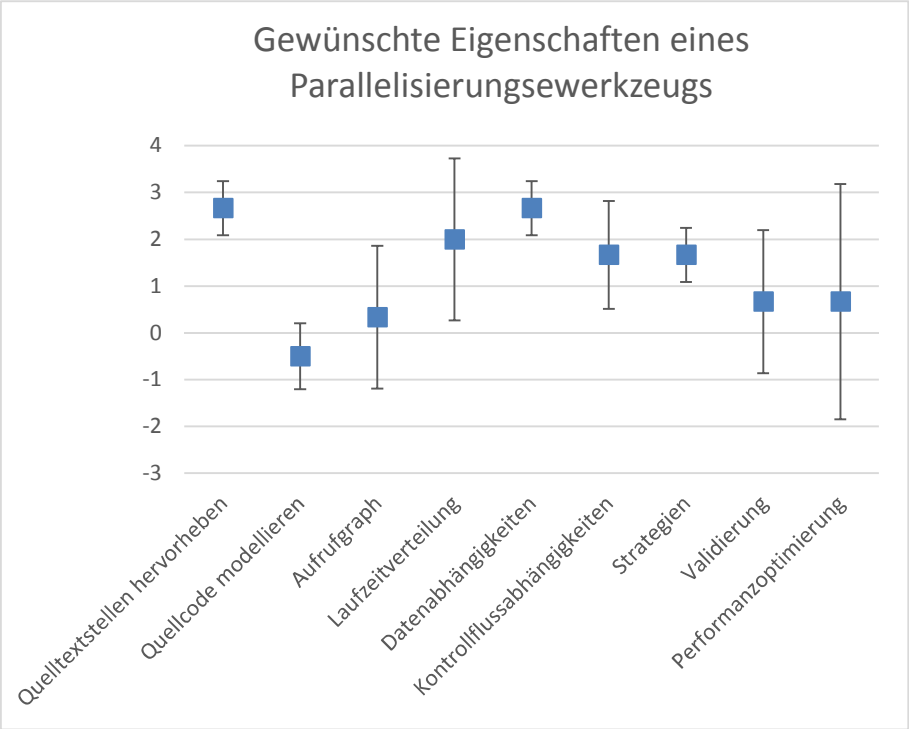

Abbildung 18: Eigenschaften, welche sich die Testteilnehmer zur Unterstützung wünschen würden

Probanden der anderen beiden Gruppen wurden ebenfalls gefragt, welchen Aspekt sie bei ihrem Werkzeug vermissen oder wo sie Verbesserungsbedarf sehen.

Bei Parallel Studio wurde in 75% der Fälle (N=3) die fehlende Unterstützung bei der Annotation bemängelt. Es wurde gewünscht, dass das Werkzeug sich nicht auf die Ausgabe der Laufzeiten beschränkt, sondern zumindest Hinweise auf parallelisierbare Stellen angibt oder sogar selbst-ständig annotiert. Was bei Parallel Studio also negativ auffällt sind Aspekte, welche Patty bereits umsetzt.

Bei Patty wurde die Darstellung des Laufzeitverhaltens gewünscht. Als Schwachpunkt wurde auch angemerkt, dass Patty nicht alle Abhängigkeiten korrekt erkannt hat, was zum derzeitigen Entwicklungsstand auch noch der Fall ist (Erklärung siehe [H13]).

6.3.3 Auswertung der Bildschirmvideos

Arbeiten mit den Werkzeugen

In Abbildung 19 werden die anhand der Bildschirmvideos gemessenen Zeitintervalle dargestellt. Aus Gründen der Übersichtlichkeit werden hier nur die Stichproben-Mittelwerte angegeben. Die vollständigen Daten befinden sich in Anhang D. Teilnehmer der mit Patty arbeitenden Testgruppe begannen sofort nach Testbeginn (0,33 [SD 0,58] Minuten) mit dem Werkzeug zu arbeiten, während Mitglieder der Parallel Studio Gruppe sich zunächst einlasen und erst nach durchschnittlich 8 [SD 9,13] Minuten begannen. Die große Streuung erklärt sich hier wieder durch die Erfahrung der Teilnehmer. Entwickler mit wenigen Kenntnissen haben deutlich mehr Zeit zum Einlesen aufgewendet. Diese Verzögerung führte dazu, dass das Auffinden der ersten parallelisierbaren Codestelle bei Patty mit 6,66 [SD 3,79] Minuten fast doppelt so schnell war wie bei Parallel Studio mit 13,5 [SD 7,05] Minuten. Die Teilnehmer der manuellen Gruppe waren nach durchschnittlich drei Minuten erstaunlich schnell beim ersten parallelisierbaren

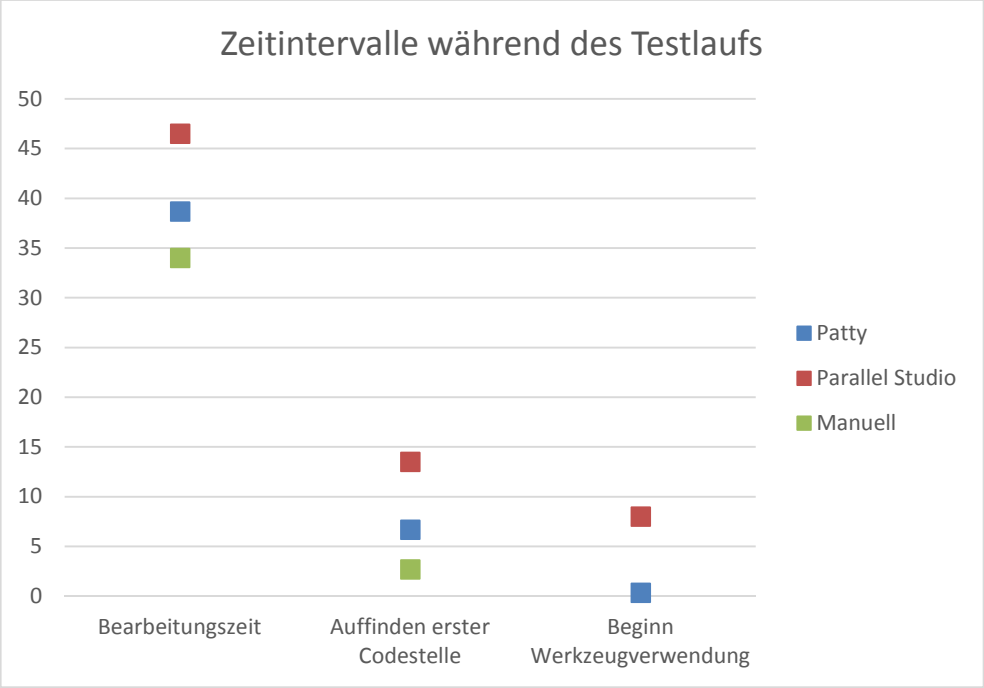

Abbildung 19: Anhand der Videoauswertung gemessene Zeitintervalle während der Testphase in Minuten

Codesegment angekommen. Dies erklärt sich dadurch, dass diese Gruppe die von Visual Studio bereitgestellten Laufzeitverteilungen zur Seite gestellt bekamen. Die dort als laufzeitintensivste ermittelte ist gleichzeitig eine mögliche parallelisierbare Methode. Die Teilnehmer der Parallel Studio Gruppe identifizieren ihre Codestellen ebenfalls anhand der Laufzeiten. Rechnet man bei dieser Gruppe die Einarbeitungszeit raus, so entdecken die Teilnehmer die gleichen Codestellen ähnlich schnell. Beiden Gruppen stehen dann jedoch keine weiteren Informationen außer den Laufzeiten zur Verfügung. Insbesondere sind keine Hinweise auf eine mögliche Parallelisierbarkeit verfügbar. Beim Arbeiten mit Patty liegen beim Auffinden der ersten Quelltextstelle bereits die Datenabhängigkeiten vor. Außerdem erhalten Patty-Nutzer die Information, ob eine Codestelle aufgrund schon durchgeführter Analysen bereits verworfen wurde. Das Fehlen dieser Informationen hatte insbesondere bei den unerfahrenen Testteilnehmern, welche mit Parallel Studio gearbeitet haben, weitere Konsequenzen. Da die Laufzeit zunächst der einzige Orientierungspunkt ist, haben die Teilnehmer viel Zeit damit verbracht, Methoden zu untersuchen, welche zwar laufzeitintensiv, jedoch aufgrund von

Datenabhängigkeiten nicht parallelisierbar sind. Die Angabe von Datenabhängigkeiten kann dies verhindern. Bei der Videoauswertung von Patty fiel außerdem auf, dass gerade unerfahrene Entwickler viel Zeit mit der Betrachtung des Abhängigkeitsgraphen verbracht haben.

In Kapitel 3.4 wurde bereits angesprochen, dass sich Parallel Studio während des Parallelisierungsprozesses auf Entscheidungen der Entwickler verlässt. Welche schwerwiegenden Folgen dies bei falschen Entscheidungen mit sich bringen kann, wurde im Verlauf der Studie klar. Ein unerfahrener Entwickler traf durch Verwendung des Werkzeugs anhand der Laufzeitanalyse auf eine Methode im Code, welche parallel ausführbar wäre, annotierte jedoch nicht die richtigen Zeilen. Da Parallel Studio die folgende Prognose nur für die vom Benutzer annotierten Bereiche durchführt, wurde das Parallelisierungspotential nicht erkannt. Der Benutzer verwarf daraufhin die Methode und untersuchte den Code auf weitere Stellen. Parallel Studio verspielt hier sein Potential, indem es sich auf die Benutzerentscheidungen einschränkt. Nach einer Stunde Bearbeitungszeit hatte der unerfahrene Entwickler lediglich eine Codestelle identifiziert. Ein unerfahrener Entwickler, welcher mit Patty gearbeitet hat, konnte in der gleichen Zeit drei Quelltextabschnitte aufdecken. Insbesondere die bei Patty dargestellten, bei Parallel Studio jedoch nicht angegebenen Datenabhängigkeiten, wurden von ihm lange Zeit betrachtet und im Fragebogen positiv hervorgehoben.

Vergleicht man hingegen zwei erfahrene Entwickler, so lässt sich in dieser Studie weder in der Anzahl der gefundenen Ergebnisse, noch in der Dauer der gesamten Bearbeitungszeit oder dem Zeitaufwand zum Auffinden der parallelisierbaren Abschnitte ein großer Unterschied feststellen.

Mit durchschnittlich 34 Minuten [SD 2,65] benötigten die Teilnehmer der manuellen Gruppe am wenigsten Zeit. Mit Patty wurde im Schnitt 38,67 [SD 14,74] Minuten, mit Parallel Studio 46,5 [SD 14,39] Minuten gearbeitet. Patty ist also 20% schneller. Wie in 6.3.2 festgestellt, wurde in der kürzeren Zeit sogar noch 36% mehr parallelisierbare Codestellen aufgedeckt.

6.4 Fazit

Insgesamt wurde Patty mit der Note 1,75 bewertet, Parallel Studio erhielt eine 2,6. Die manuelle Testgruppe schnitt klar am schlechtesten ab. Zwar beendeten die Testpersonen hier am schnellsten ihre Arbeit, sie hatten aber die wenigsten Quelltextstellen korrekt identifiziert. Zudem war sie die einzige Gruppe, welche falschpositive Ergebnisse ermittelte. Trotzdem sind sie am zufriedensten mit ihren Ergebnissen. Ihnen war also nicht klar, dass ihre Ergebnisse unvollständig und fehlerhaft sind. Manuelle Parallelisierung ist also unsauber. Es ist demnach sinnvoll, Entwickler bei der Parallelisierung durch Werkzeuge zu unterstützen.

Bei der Notenverteilung von Parallel Studio fällt auf, dass eine sehr gute Note nur von fortgeschrittenen Anwendern vergeben wurde. Parallel Studio ist ein kommerzielles Produkt und ist durch zahlreiche Analysewerkzeuge und eine Vielzahl an Details deutlich umfangreicher als Patty. Dieser vermeintliche Pluspunkt hat jedoch den Nachteil, dass Entwickler ohne großartige Kenntnisse mit dem Werkzeug kaum zurechtkommen und sogar grobe Fehler machen. Fortgeschrittene Anwender kommen mit dem Werkzeug hingegen schnell ans Ziel und empfinden das Arbeiten als angenehm und einfach. Trotz der großen Menge an Analysemöglichkeiten und Informationen, werden dem Benutzer bei seiner Suche nach Parallelisierungspotential ausgerechnet die Datenabhängigkeiten oder aber auch sonstige Hinweise nicht angezeigt. Den Entwicklern bleiben nur die Laufzeiten zur Orientierung. Als Folge wird von den Entwicklern viel Zeit an Codestellen verbracht, welche für eine parallele Ausführung gar nicht geeignet sind. Im schlimmsten Fall kann es sogar, wie auch in der Studie geschehen, dazu kommen, dass ein Entwickler Parallelisierungspotential entdeckt, jedoch durch falsche Annotation nicht in parallelen Quellcode umsetzten kann.

Bei Patty ist das Gegenteil der Fall. Die Noten sowie die Ergebnisse von unerfahrenen Entwicklern fallen deutlich besser aus. Schlechte Bewertungen erfolgen nur durch erfahrene Entwickler, welche die Schwächen der noch prototypischen Entwicklung von Patty erkennen und beanstanden.

Insgesamt geht Patty aus der Studie als beste Möglichkeit zur Parallelisierung serieller Quelltexte hervor. Durchschnittlich wurden die meisten Codestellen identifiziert (Kapitel 6.3.2) und die wenigsten Fehler gemacht (Kapitel 6.3.3). Das Arbeiten mit Patty war schneller (Kapitel 6.3.3) und somit auch effizienter als das Arbeiten mit Parallel Studio. Nur die manuelle Gruppe war etwas schneller (Kapitel 6.3.3), hatte dafür aber deutlich schlechtere Ergebnisse (Kapitel 6.3.2).

6.4.1 Erkenntnisse für Patty

Patty bietet die von der manuellen Testgruppe am meisten gewünschten Merkmale: die Visualisierung von Datenabhängigkeiten und das Hervorheben von Parallelisierungspotential. Die geringe Streuung bei diesen beiden Merkmalen (jeweils 0,58) bestätigt weiter, dass diese Programmfunktionen allgemein als sehr hilfreich empfunden werden. Patty kommt somit genau dem nach, was Entwickler für eine Parallelisierung benötigen

Der Test hat schon noch vorhandene Schwachstellen von Patty bestätigt und auch einige Verbesserungsmöglichkeiten aufgedeckt. Was oft als hilfreich empfunden (Wertung der manuellen Gruppe 2 [SD 1,73], sowie von 33% (N=1) der Patty-Gruppenteilnehmer genannt), aber von Patty noch nicht angeboten wird, ist die Berechnung des Rechenaufwands des zu parallelisierenden Codes. Um diese Funktionalität sollte Patty im Rahmen der Dynamischen Analyse erweitert werden. Eine Erweiterung um das manuelle Hinzufügen von Abhängigkeiten, wie es von den Testpersonen angemerkt wurde, ist sinnvoll, da nicht garantiert werden kann, dass Patty stets alle Abhängigkeiten selbst findet. Dies kann immer dann passieren, wenn der Quellcode einer referenzierten Bibliothek nicht zur Verfügung steht.

In der Auswertung der Fragebögen wurde die Nachvollziehbarkeit durch die Benutzer sehr positiv bewertet, womit das in der Konzeption erklärte Ziel (Kapitel 4.3.2) erreicht wurde.

7 ZUSAMMENFASSUNG UND AUSBLICK

Ziel dieser Arbeit war das Erstellen und insbesondere Bewerten einer Benutzerschnittstelle zur automatischen Parallelisierung von seriellen Quelltexten. Zugrunde lag der Arbeit das von der Arbeitsgruppe AParT entwickelte Parallelisierungswerkzeug Patty.

Software parallel zu entwickeln ist schwieriger und fehlerbehafteter als die Entwicklung serieller Programme. Aufgrund der technischen Grenzen müssen Softwareprodukte jedoch angepasst werden, um den steigenden Rechenbedarf zu decken.

Um die Akzeptanz sowie den Einstieg in die automatisierte Parallelisierung von Quelltexten zu erleichtern, war das oberste Ziel bei der Erstellung der Schnittstelle stets, diese möglichst nachvollziehbar und insbesondere für unerfahrene Entwickler einfach verwendbar zu gestalten. Hierfür wurden aus anderen Arbeiten Konzepte wie die Darstellung von Abhängigkeiten in einem Graphen oder die Einbindung in eine existierende Entwicklungsumgebung übernommen.

Im Rahmen der Evaluation wurde eine Entwicklerstudie durchgeführt. In dieser Studie trat Patty als Parallelisierungswerkzeug gegen das Intel Parallel Studio sowie gegen eine Testgruppe ohne Werkzeugunterstützung an.

Ein Ergebnis der Studie ist, dass das Arbeiten mit Werkzeugen im Vergleich zur Parallelisierung ohne Werkzeuge weniger fehlerbelastet, gründlicher und schneller ist. Es ist demnach sinnvoll, Werkzeuge zur Unterstützung von Softwareentwicklern in diesem Bereich zu entwickeln und zu verbessern.

Patty kann als prototypische Entwicklung natürlich nicht mit dem Leistungsumfang von Parallel Studio mithalten. Dennoch trat Patty in der Studie als Sieger sowohl in der durchschnittlichen Gesamtwertung, als auch beim selbstgesteckten Ziel der Nachvollziehbarkeit, hervor. Es hat 36% mehr parallelisierbare Codestellen aufgedeckt als Parallel Studio und 50% als eine manuelle Untersuchung des Programmcodes. Weiter wurde gezeigt, dass die in der Konzeption ermittelten Wege tatsächlich eine große Hilfe für Einsteiger und unerfahrene Entwickler sind. Es wurde sogar festgestellt, dass das Weglassen mancher dieser Informationen, welche von Patty im Gegensatz zu Parallel Studio angeboten werden, große Folgen mit sich bringt. Ohne die Abhängigkeitsgraphen wurden Abhängigkeiten, welche auch Datenkonflikte nach sich ziehen, nicht immer erkannt. Fehlende Hinweise an den Entwickler führten sogar dazu, dass im Verlauf des Parallelisierungsvorgangs gültige Quelltextsegmente von unerfahrenen Entwicklern verworfen wurden.

Die durchgeführte Studie ist mit zehn Testpersonen natürlich nicht repräsentativ. Eine zukünftige, umfangreichere Studie mit einer großen Anzahl an Probanden, würde noch weiter Aufschlüsse bringen.

Im Zuge dieser Arbeit wurde gezeigt, dass nicht nur die hinter Patty stehende Intention, nämlich die Automatisierung der Parallelisierung sinnvoll, sondern auch die Art und Weise, wie diese umgesetzt wird, zielführend ist.

ANHÄNGE

A. Abkürzungsverzeichnis

MSDN	**Microsoft Developer Network**
DGML	**Directed Graph Markup Language**
IDE	**Integrated Development Environment**
TADL	**Tunable Architecture Description Language**
UEQ	**User Experience Questionnaire**
GUI	**Graphical User Interface**
KFG	**Kontrollflussgraph**
UX	**User Experience**
MW	**Mittelwert**
SD	**Standard Deviation**
MS	**Microsoft**
CPU	**Central Processing Unit**
VAR	**Varianz**

B. Abbildungsverzeichnis

C. Evaluierungsmaterial

Fragebögen: Gemeinsame Elemente

1 Über diesen Fragebogen

Lieber Untersuchungsteilnehmer,

mit dem vorliegenden Fragebogen sollen drei Herangehensweisen einer Parallelisierung von seriellem Quelltext beurteilt werden.
Diese Herangehensweisen sind einmal manuell sowie softwareunterstützt mit zwei verschiedenen Werkzeugen.
Anhand des Fragebogens sollen Stärken und Schwächen der jeweiligen Vorgehensweisen ermittelt werden.
Außerdem soll die Benutzbarkeit der Programme mit einer grafischen Benutzeroberfläche festgestellt werden.

Vergessen sie bitte nicht, Ihren Namen auf dem Fragebogen zu notieren.

Der Fragebogen enthält drei verschiedene Fragetypen

1.1 Entscheidungsfragen

Entscheiden Sie für eine gegebene Aussage, ob diese für Sie zutrifft oder nicht, indem sie entsprechend ankreuzen.

Beispiel:
Haben Sie die Aufgabenstellung verstanden? ☐ Ja ☐ Nein

1.2 Wertungsskala

Schätzen Sie das Produkt anhand der angegebenen Skalen ein. Die Skalen bestehen jeweils aus Gegensatzpaaren von Eigenschaften, welche das Produkt haben kann.

Beispiel:

attraktiv	○	⊗	○	○	○	○	○	unattraktiv

Mit dieser Beurteilung sagen Sie aus, dass Sie das Produkt eher attraktiv als unattraktiv einschätzen.
Kreuzen Sie bitte nur einen Kreis pro Zeile an.

1.3 Freitext

Bei Freitextfragen steht ihnen ein Textfeld zur Verfügung, in welches Sie Ihre Antwort eintragen können.

2 Fragen

2.1 Subjektive Einschätzungen

Zunächst werden Ihnen Fragen zu Ihren Erfahrungen im Bereich Softwareentwicklung gestellt. Darauf folgen Fragen, welche Ihr Verständnis der Aufgabenstellung abfragen.

1. **Wie viele Jahre Erfahrung haben Sie als Software-Entwickler?**

---

2. Wie viele Jahre Erfahrung haben Sie im Umgang mit objektorientierter Entwicklung?

---

3. Wie viel Erfahrung haben Sie im Bereich Multicore?

---

4. Wie gut haben Sie die Aufgabenstellung verstanden?

---

5. Wie gut kamen sie mit der Ihnen gestellten Aufgabe zurecht?

	1	2	3	4	5	6	7	
gut	○	○	○	○	○	○	○	schlecht

6. Die Aufgabenstellung hat mich…

	1	2	3	4	5	6	7	
unterfordert	○	○	○	○	○	○	○	überfordert

7. Beurteilen sie die Anforderungen, welche die Aufgabe an sie stellt.

	1	2	3	4	5	6	7	
niedrig	○	○	○	○	○	○	○	hoch

8. Wie zufrieden sind Sie mit Ihrem Ergebnis?

	1	2	3	4	5	6	7	
sehr	○	○	○	○	○	○	○	gar nicht

Ergebnisse

Bitte notieren Sie hier alle Codestellen, in denen sie Parallelisierungspotential vermuten

Klassenname	Methodenname	Beginn und Ende der Codestelle

Fragebögen: Spezifischer Fragenteil Werkzeugunterstützung

Werkzeugeigenschaften

Beurteilen Sie die Eigenschaften des Produkts. Entscheiden Sie möglichst spontan. Es ist wichtig, dass Sie nicht lange über die Begriffe nachdenken, damit Ihre unmittelbare Einschätzung zum Tragen kommt.

Bitte bewerten Sie den Umgang mit dem Werkzeug in den 16 angegebenen Kategorien.

	1	2	3	4	5	6	7		
unverständlich	○	○	○	○	○	○	○	verständlich	1
leicht zu lernen	○	○	○	○	○	○	○	schwer zu lernen	2
wertvoll	○	○	○	○	○	○	○	minderwertig	3
langweilig	○	○	○	○	○	○	○	spannend	4
uninteressant	○	○	○	○	○	○	○	interessant	5
unberechenbar	○	○	○	○	○	○	○	voraussagbar	6
schnell	○	○	○	○	○	○	○	langsam	7
behindernd	○	○	○	○	○	○	○	unterstützend	8
kompliziert	○	○	○	○	○	○	○	einfach	9
sicher	○	○	○	○	○	○	○	unsicher	10
aktivierend	○	○	○	○	○	○	○	einschläfernd	11
erwartungskonform	○	○	○	○	○	○	○	nicht erwartungskonform	12
ineffizient	○	○	○	○	○	○	○	effizient	13
übersichtlich	○	○	○	○	○	○	○	verwirrend	14
unpragmatisch	○	○	○	○	○	○	○	pragmatisch	15
aufgeräumt	○	○	○	○	○	○	○	überladen	16

Wie gut hat sie das zur Verfügung gestellte Werkzeug bei der Parallelisierung unterstützt?

	1	2	3	4	5	6	7	
sehr	○	○	○	○	○	○	○	gar nicht

Welchen Aspekt des Werkzeugs halten Sie persönlich für den Besten und warum?

Welchen Aspekt des Werkzeugs muss Ihrer Meinung nach verbessert werden und warum?

Fehlt dem Werkzeug Ihrer Meinung nach eine Funktionalität? Wenn ja, um welche Funktionalität sollte das Werkzeug unbedingt erweitert werden?

Abschließend: Mit welcher Gesamtnote würden sie das System bewerten?

Fragebögen: Spezifischer Fragenteil manuell

Gewünschte Eigenschaften eines Parallelisierungswerkzeugs

Ihre Testgruppe hat die Aufgabenstellung ohne Unterstützung eines Parallelisierungswerkzeugs bearbeitet. Angenommen, Sie müssten diese Aufgabe erneut bearbeiten.
Würde es Ihnen helfen, wenn sie ein Werkzeug hätten, welches…
(1 entspricht sehr hilfreich, 7 entspricht nicht hilfreich)

	1	2	3	4	5	6	7
…Quelltextstellen hervorhebt, die Parallelisierungspotential enthalten?	○	○	○	○	○	○	○
…den Quellcode modelliert?	○	○	○	○	○	○	○
…den Aufrufgraphen erstellt und visualisiert?	○	○	○	○	○	○	○
…die Laufzeitverteilungen darstellt?	○	○	○	○	○	○	○
…Ihnen die Datenabhängigkeiten zwischen den Programmvariablen anzeigt?	○	○	○	○	○	○	○
…Ihnen die Kontrollflussabhängigkeiten zwischen den Programmanweisungen anzeigt?	○	○	○	○	○	○	○
…Ihnen Strategien zum möglichen Umsetzung von Parallelisierung macht?	○	○	○	○	○	○	○
…Ihnen bei der Validierung der Korrektheit des parallelen Programms hilft?	○	○	○	○	○	○	○
…Ihnen bei der Performanzoptimierung der parallelen Version hilft?	○	○	○	○	○	○	○

Gibt es noch etwas, das Ihnen bei helfen würde, aber von den bisherigen Fragen nicht erfasst wurde?

D. Auswertungen der Evaluierung

Auswertung der Wertungsskalen der Testgruppe Patty

1	7	Kategorie				MW	VAR	SD
unverständlich	verständlich	Durchschaubarkeit	2	3	2	2,33	0,33	0,58
leicht zu lernen	schwer zu lernen	Durchschaubarkeit	3	3	1	2,33	1,33	1,15
wertvoll	minderwertig	Stimulation	3	0	2	1,67	2,33	1,53
langweilig	spannend	Stimulation	3	0	-1	0,67	4,33	2,08
uninteressant	interessant	Stimulation	3	0	-1	0,67	4,33	2,08
unberechenbar	voraussagbar	Steuerbarkeit	0	2	0	0,67	1,33	1,15
schnell	langsam	Effizienz	2	3	2	2,33	0,33	0,58
behindernd	unterstützend	Steuerbarkeit	2	0	2	1,33	1,33	1,15
kompliziert	einfach	Durchschaubarkeit	3	1	2	2,00	1,00	1,00
sicher	unsicher	Steuerbarkeit	0	0	0	0,00	0,00	0,00
aktivierend	einschläfernd	Stimulation	0	0	0	0,00	0,00	0,00
erwartungskonform	nicht erwartungskonform	Steuerbarkeit	1	2	0	1,00	1,00	1,00
ineffizient	effizient	Effizienz	2	2	0	1,33	1,33	1,15
übersichtlich	verwirrend	Durchschaubarkeit	2	2	2	2,00	0,00	0,00
unpragmatisch	pragmatisch	Effizienz		0	0	0,00	0,00	0,00
aufgeräumt	überladen	Effizienz	2	1	2	1,67	0,33	0,58

Auswertung der Wertungsskalen der Testgruppe Parallel Studio

1	7	Kategorie					MW	VAR	SD
unverständlich	verständlich	Durchschaubarkeit	2	2	-1	1	1,00	2,00	1,41
leicht zu lernen	schwer zu lernen	Durchschaubarkeit	2	3	-1	1	1,25	2,92	1,71
wertvoll	minderwertig	Stimulation	3	3	1	0	1,75	2,25	1,50
langweilig	spannend	Stimulation	0	1	2	1	1,00	0,67	0,82
uninteressant	interessant	Stimulation	1	1	2	1	1,25	0,25	0,50
unberechenbar	voraussagbar	Steuerbarkeit	3		2	-1	1,33	4,33	2,08
schnell	langsam	Effizienz	2	3	3	2	2,50	0,33	0,58
behindernd	unterstützend	Steuerbarkeit	3	3	2	2	2,50	0,33	0,58
kompliziert	einfach	Durchschaubarkeit	3	-1	0	1	0,75	2,92	1,71
sicher	unsicher	Steuerbarkeit	2	2	2	1	1,75	0,25	0,50
aktivierend	einschläfernd	Stimulation	2	0	0	0	0,50	1,00	1,00
erwartungskonform	nicht erwartungskonform	Steuerbarkeit	3	2	1	1	1,75	0,92	0,96
ineffizient	effizient	Effizienz	2	0	2	-2	0,50	3,67	1,91
übersichtlich	verwirrend	Durchschaubarkeit	3	2	-2	1	1,00	4,67	2,16
unpragmatisch	pragmatisch	Effizienz	3	2	-1	1	1,25	2,92	1,71
aufgeräumt	überladen	Effizienz	3	2	-2	1	1,00	4,67	2,16

Auswertung der Wertungsskalen der manuellen Testgruppe

				MW	VAR	SD
Quelltextstellen hervorheben	2	3	3	2,66666667	0,33333333	0,57735027
Quellcode modellieren	-1		0	-0,5	0,5	0,70710678
Aufrufgraph	-1	0	2	0,33333333	2,33333333	1,52752523
Laufzeitverteilung	0	3	3	2	3	1,73205081
Datenabhängigkeiten	2	3	3	2,66666667	0,33333333	0,57735027
Kontrollflussabhängigkeiten	1	1	3	1,66666667	1,33333333	1,15470054
Strategien	2	1	2	1,66666667	0,33333333	0,57735027
Validierung	-1	1	2	0,66666667	2,33333333	1,52752523
Performanzoptimierung	-2	1	3	0,66666667	6,33333333	2,51661148

Auswertung der Zufriedenheit

Transformierte Angaben der Zufriedenheit von sehr bis gar nicht und Auswertung:

Patty	*Patty*	*Patty*	*PS*	*PS*	*PS*	*PS*	*Man*	*Man*	*Man*
0	1	1	1	-2	3	-3	0	2	1

	MW	*Varianz*	*SD*
Patty	0,666666667	0,333333333	0,577350269
Parallel Studio	-0,25	7,583333333	2,753785274
Manuell	1	1	1

Auswertung der Gesamtnote

Benotung Patty:

Vergebene Noten		*MW*	*VAR*	*SD*
1,5	2	1,75	0,125	0,35355339

Benotung Parallel Studio:

Vergebene Noten				*MW*	*VAR*	*SD*
3	3	2,7	1,7	2,6	0,38	0,6164414

Empfundenen Unterstützung bei der Parallelisierung

Auswertung der Bildschirmvideos

Zeitintervalle in Minuten:

	Patty	*Patty*	*Patty*	*PS*	*PS*	*PS*	*PS*	*Manuell*	*Manuell*	*Manuell*
Gesamte Bearbeitungszeit	50	44	22	30	60	39	57	35	31	36
Dauer bis Auffinden erster Codestelle	11	4	5	6	19	9	20	5	1	2
Zeitdauer bis Beginn Werkzeugverwendung	1	0	0	1	12	0	19			

Gesamte Bearbeitungszeit in Minuten:

	MW	*Varianz*	*SD*
Patty	38,6666667	217,333333	14,7422296

Parallel Studio	46,5	207	14,3874946
Manuell	34	7	2,64575131

Auffinden der ersten Codestelle in Minuten:

	MW	*Varianz*	*SD*
Patty	6,66666667	14,3333333	3,7859389
Parallel Studio	13,5	49,6666667	7,04745817
Manuell	2,66666667	4,33333333	2,081666

Beginn der Werkzeuganwendung in Minuten:

	MW	*Varianz*	*SD*
Patty	0,33333333	0,33333333	0,57735027
Parallel Studio	8	83,3333333	9,12870929

E. Literaturverzeichnis

[FM11] Samuel H. Fuller, Lynette I. Millett: *The Future of Computing Performance: Game Over or Next Level?*, 2011, The National Academies Press

[BF10a] Ilona Bluemke, Joanna Fugas: *A Tool Supporting C code Parallelization* in *Innovations in Computing Sciences and Software Engineering*; Seite 259 - 264, Springer, 2010

[BF10b] Ilona Bluemke, Joanna Fugas: *C code parallelization with ParaGraph* in *Proceedings*: *2nd International Conference on Information Technology, ICIT 2010*, Pages 163-166, 2010

[GF99] Maurizio Giordano, Mario M. Furnari: *HTGviz Graphic Tool for the Synthesis of Automatic and User-Driven Program Parallelization in the Compilation Process* in Proceedings: *High Performance Computing*, Seite 312-319, Springer, 1999

[INT] Intel, *https://software.intel.com/en-us/sites/default/files/Intel_Parallel_Studio_XE_2013_PB-101512.pdf*, 2012

[CRI] CriticalBlue YouTube Channel: *http://www.youtube.com/channel/UC1pK4eot50xGZ27wkEREbqw*

[OPE] OpenMP 4.0 Specification, 2013

[BB+09] Hansang Bae, Leonardo Bachega: *Cetus: A Source-to-Source Compiler Infrastructure for Multicores* in *Compute,* Volume 4, Seite 36-42

[SWT13] Oliver Hummel: *Softwaretechnik II*, Karlsruher Institut für Technologie (KIT), Topic 12, 2013

[SB11] Florian Sarodnick, Henning Brau: *Methoden der Usability Evaluation*, 2. Auflage, Huber, 2011

[H13] Jochen Huck, *Generierung paralleler Architekturbeschreibungen durch kombinierte statische und dynamische Analysen*, Diplomarbeit, Karlsruher Institut für Technologie (KIT), 2013

[ES13] Karl Eilebrecht, Gernot Starke: *Patterns kompakt: Entwurfsmuster für effektive Softwareentwicklung*, 4. Auflage, Springer, 2013

[DGML] MSDN, *Grundlegendes zur Directed Graph Markup Language (DGML)*,

http://msdn.microsoft.com/de-de/library/ee842619.aspx#DGML, Stand 15.09.2014

[XML] MSDN, *XmlTextWriter-Klasse*, http://msdn.microsoft.com/de-de/library/system.xml.xmltextwriter%28v=vs.110%29.aspx, Stand 15.09.2014

[UEQ] http://www.ueq-online.org/index.php/what-is-the-user-experience-questionnaire/?lang=de, Stand 15.09.2014

[HK10] Norbert Henze, Dieter Kadelka: *Wahrscheinlichkeitstheorie und Statistik für Studierende der Informatik und des Ingenieurwesens*, 2010

[RO12] Rüdiger Dillmann, Tamim Asfour, *Pipeline-Verarbeitung*, Karlsruher Institut für Technologie (KIT),Vorlesung Rechnerorganisation Kapitel 5, 2012

[W13] Simon Wagner, *Automatische Parallelisierung sequenzieller Programme mittels Architekturbeschreibungen*, Diplomarbeit, Karlsruher Institut für Technologie (KIT), 2013

[BAC] MSDN, *BackgroundWorker-Klasse*, http://msdn.microsoft.com/de-de/library/system.componentmodel.backgroundworker%28v=vs.110%29.aspx, Stand 15.09.2014

[STS] Wikipedia, http://en.wikipedia.org/wiki/Source-to-source_compiler, Stand 15.09.2014