Visualisierung der Datenstrukturen des Hardware/Software Co–Entwurf Systems COSYMA

J.Henkel, H. Grosenick, P. Lüders, Th. Benner, R. Ernst

Technische Universität Braunschweig
Institut für Datenverarbeitungsanlagen
Hans–Sommer–Str. 66
D–38106 Braunschweig
Email: henkel@ida.ing.tu-bs.de
Fax: 0531/391-4587
☏ 0531/391-3728

(gefördert durch die DFG)

Abstrakt

Es wird eine Visualisierungsmöglichkeit zur Darstellung der internen Datenstrukturen des Experimentalsystems COSYMA für den Hardware-/Software Co–Entwurf vorgestellt. Die graphische Darstellungsmöglichkeit wurde aufgrund der Komplexität der internen Graphstruktur für den Zweck des Debugging sowie als Interface zum benutzerunterstützten Co–Entwurf notwendig.
Als Visualisierungssystem wurde GraphADL gewählt [LE92], weil es die Darstellung hierarchischer Graphen erlaubt und bereits erfolgreich für andere Anwendungen eingesetzt wurde.

1 Einleitung

Das Anwendungsgebiet für integrierte Steuerungssysteme (*embedded control*), die nur aus wenigen Bauteilen bestehen, expandiert ständig. Einsatzbereiche wie Automobiltechnik, Büroautomatisierung, Telekommunikation etc. profitieren davon, indem die Funktionalität erhöht wird bei gleichzeitiger Senkung der Kosten.
Das sogenannte Gebiet des Hardware/Software Co-Entwurfs übernimmt in dieser Entwicklung eine Schlüsselrolle: ein Echtzeitsystem wird lediglich durch sein Verhalten beschrieben, die Formulierung zeitlicher Randbedingungen garantiert die Einhaltung zeitkritischer Aufgaben in einem Echtzeitsystem.
Ziel des Co-Entwurfs ist die späte Trennung von Hardware- und Softwareanteilen im Entwurf.
Als Fernziel formuliert der Systementwickler nur noch die Spezifikation, eine Hardware/Software–Partitionierung erstellt das System weitgehend automatisch aus den vorgegebenen Randbedingungen (Zeitrestriktionen, Leistungsbedarf, Chipfläche,...). Hardware- und Softwarebeschreibungen werden dann mit vorhandenen Verfahren wie z.B. High–Level Synthese weiterverarbeitet. Der Vorteil einer derartigen Vorgehensweise liegt in der erheblichen Reduzierung der Entwurfszykluszeiten (*turnaround time*), da möglichst viele Entwurfsschritte automatisiert werden .

Als Zielarchitektur kommen derzeit nur kleine

Systeme — integrierte Systeme — in Frage, da benötigte Algorithmen und Verfahren (z.B. High–Level Synthese) für komplexere Systeme noch zu rechenintensiv sind.
Derzeitige Trends unterstützen die Forschungsaktivitäten im Gebiet des Hardware/Software Co–Entwurfs:

- immer mehr Mikrocontroller werden in der Programmiersprache C programmiert. Die Partitionierung als einer der Kernpunkte des Hardware/Software Co–Entwurfs ist damit unabhängig vom Befehlssatz des Mikrocontrollers, findet also auf algorithmischer Basis statt.
- viele Forschungsgebiete, die für den Hardware/Software Co–Entwurf von Wichtigkeit sind (High–Level Synthese, Hardware-/Software Co–Simulation,...) haben neuerdings gute Fortschritte erzielen können.
- Die Rechenleistung heutiger Workstations ist zumindest teilweise in der Lage, anstehende komplexe Optimierungsaufgaben in akzeptablen Zeitrahmen zu berechnen.

Unser Ansatz zum Hardware/Software Co–Entwurf konzentriert sich auf die Fragestellung der Automatisierung der Hardware/Software Partitionierung.

Heute existieren nur sehr wenige Ansätze in dem Gebiet des Hardware/Software Co–Entwurfs ([GuMi92], [BaRo92]).
Das VULCAN System ([GuMi92] und unser COSYMA (**COSY**nthesis for e**M**bedded **A**rchitecture) System nehmen eine automatische Partitionierung unter zeitlichen Randbedingungen und Optimierungskriterien vor.
Sowohl VULCAN als auch COSYMA besitzen dieselbe Zielarchitektur: kleine eingebettete Systeme wie Mikrocontroller, die aus einem vordefinierten Standardprozessorkern und einem applikationsspezifisch generierten Coprozessor sowie Speicher und Peripheriekomponenten bestehen. Beide Ansätze iterieren über den Synthese– und Kompilationsprozeß, weil es derzeit keine brauchbaren Verfahren gibt, die genau und in kurzer Zeit die Ergebnisse optimierender Compiler und der High–Level Synthese (mit neuen Techniken wie *loop unfolding, tree height reduction*) schätzen können.

Nach der Vorstellung der Werkzeuge des Hardware/Software Co–Entwurf Systems COSYMA im nächsten Kapitel folgt in Kapitel 3 die Beschreibung der internen Datenstrukturen. Daran schließt sich die Beschreibung der Visualisierungsmöglichkeiten und benutzergestützten Steuerung der einzelnen Werkzeuge von COSYMA an. Schließlich gibt Kapitel 5 eine Zusammenfassung.

2 Hardware/Software Co–Entwurf mit COSYMA

In Abbildung 1 ist der vereinfachte Entwurfsablauf des COSYMA–Systems zu sehen: der Entwickler beschreibt ein Echtzeitsystem in der Sprache C^x, eine Obermenge der Sprache ANSI C, die um folgende Konstrukte erweitert wurde:

- Formulierung zeitlicher Randbedingungen
- Parallelität
- *Assertions* zur Unterstützung der Laufzeitanalyse

Daraufhin wird mit Hilfe des C^x–Compilers das interne Datenformat, der sogenannte **E**rweiterte **S**yntax–Graph (ES-Graph) generiert. Hierauf wird die automatisierte kostenfunktionsgesteuerte Partitionierung[1] vorgenommen,

[1] Eine Beschreibung der Partitionierungsalgorithmen kann in [HBE+93] nachgelesen werden.

die nach Hardware auszulagernde Systemteile im Graphen markiert. Dies ist die sogenannte innere Partitionierungsschleife.
Dann wird aus diesen Graphteilen eine Hardwarebeschreibung generiert, woraus mit Hilfe eines High–Level Synthese Systems eine applikationsspezifische Hardware generiert wird. Alle anderen Teile des Graphen werden in ein C–Programm zurückübersetzt und mit einem Standard C-Compiler (GNU) kompiliert, nachdem Kommunikationsfunktionen eingefügt wurden, die die Software/Hardware und Hardware-/Software Kommunikation bewerkstelligen.

Nachdem die Hardwareausführungszeiten der applikationsspezifischen Hardware durch Simulation gewonnen wurden, werden sie in den Assemblercode in codierter Form eingebaut, so daß eine Laufzeitanalyse für Hardware– und Softwareteile gemeinsam vorgenommen werden kann[2]. Wird in der äußeren Schleife, bestehend aus Softwaregenerierung, Hardwaregenerierung und Laufzeitanalyse, festgestellt, daß zeitliche Randbedingungen nicht eingehalten werden können, muß über die Partitionierung mit modifizierten Kostenparametern iteriert werden.
Detaillierter wird COSYMA in [ErHeBe93] beschrieben.
Probleme, die auf den Kommunikationsoverhead zwischen Hardware und Software zurückzuführen sind, werden in [HeBeEr93] diskutiert.

[2]Hierfür existiert ein Werkzeug, das eine Laufzeitanalyse auf Basis eines Standardprozessorkerns (zur Zeit SPARC) mit Berücksichtigung von Pipelineeffekten, Datenabhängigkeiten etc. ausführt ([YErBeHe93]).

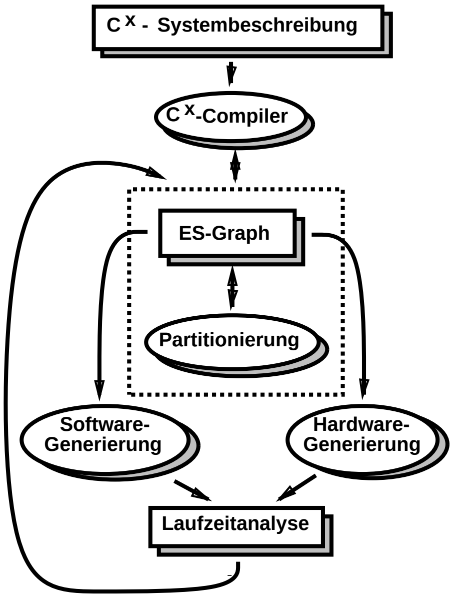

Abbildung 1: Vereinfachter Entwurfsablauf des COSYMA-Systems

3 Datenstrukturen

Üblicherweise werden im Bereich des High–Level Entwurfs Datenflußgraphen (DFG) oder Steuer– und Datenflußgraphen (CDFG) als interne Repräsentation gewählt.
In [HEH+93] wird ein Syntheseformat benutzt, das einen Schritt weiter geht und die Repräsentation eines Systems auf verschiedenen Abstraktionsniveaus erlaubt.

Während aus DFG bzw. aus CDFG sehr einfach die Datenabhängigkeiten hervorgehen, ist die Darstellung von dynamischen Datenstrukturen kaum möglich.
Eine Syntaxgraph–Darstellung kann den gesamten Sprachumfang darstellen, und kann sehr einfach ohne Strukturverlust in die Ausgangssprache zurückübersetzt werden, aber Datenabhängigkeiten sind daraus nur sehr schwer abzuleiten.
Um die Vorteile beider Darstellungsarten in COSYMA nutzen zu können, wurde der soge-

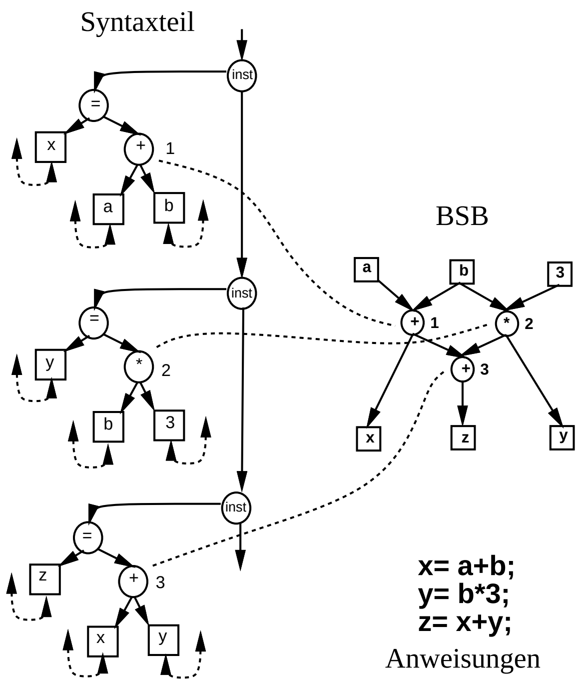

Abbildung 2: Syntaxgraphteil und Datenflußgraphteil des ESG

nannte **E**rweiterte **S**yntax **G**raph (ESG) entwickelt. Es handelt sich dabei um einen gerichteten azyklischen Graphen, der eine Folge von Deklarationen, Definitionen und Anweisungen enthält. Jeder im Graph vorkommende Bezeichner enthält einen Verweis auf die zugehörige Definitionsstelle. Dort befinden sich Zeiger auf sämtliche Benutzungsstellen.
Abbildung 2 zeigt ein kurzes Codesegment, das aus drei einfachen Anweisungen besteht. Eine Definition wird durch einen Definitions- (*def*) und eine Anweisung durch einen Instruktionsknoten (*inst*) eingeleitet.

Die gestrichelten Pfeile entsprechen den besagten Verweisen. Diese implizite Symboltabelle gibt aber noch keine Auskunft über die Datenabhängigkeiten zwischen den Operationen.
COSYMA benötigt dies aber einerseits um Abschätzungen über den Software/Hardware– und Hardware/Software–Kommunikationsaufwand berechnen zu können und um ein lokales Scheduling ausführen zu können, das Auskunft über mögliche Geschwindigkeitsvorteile im Falle der Hardwareimplementierung gibt.
Die dafür notwendigen Datenabhängigkeiten werden durch einen zweiten Graphteil dargestellt, der aus miteinander verketteten Blöcken — den sogenannten **B***asic* **S***cheduling* **B***locks* (BSB) — besteht, mit denen der Kommunikationsaufwand bestimmt und die potentielle Parallelität in Hardware durch ein lokales Scheduling geschätzt wird. Die BSBs enthalten lokale Datenflußgraphen (einer davon ist auf der rechten Seite von Abbildung 2 zu sehen).
Beide Graphteile besitzen identische Operatorknoten, so daß ein schneller Übergang möglich ist, was Rechenzeit beim Partitionierungsprozeß einspart.

4 Visualisierung des ES–Graphen

Die Visualisierung des ES–Graphen wurde aus folgenden Gründen notwendig:

- Während der frühen Projektphasen mußte eine Möglichkeit geschaffen werden, Fehler zu lokalisieren.
  Die Möglichkeit, den ES–Graphen in ASCII Form zu speichern (CGL-Format genannt), konnte dabei nur bedingt genutzt werden, da das entsprechende Format sehr umfangreich und unübersichtlich war: z.B. benötigen 1400 Zeilen C-Quellcode ca. 800 kB im CGL-Format.

- Neben der automatisierten Partitionierung in COSYMA bestand der Bedarf nach manueller Partitionierung für Testzwecke. Hierzu mußte eine Instrumentarium geschaffen werden, von verschiedenen Hierarchiestufen (BSBs, Schleifenkonstrukte, Funktionen) des Graphen aus Markierungen vornehmen zu können.

- Schließlich sollte eine Möglichkeit gefunden werden, die Parameter für den Co–Entwurf interaktiv auswählen zu können sowie den Aufruf der einzelnen Werkzeuge möglichst einfach zu gestalten.

Zu diesem Zweck wurde GraphADL ([LE92]), ein Hilfsmittel zum Edieren und zur Analyse von komplexen Graphstrukturen, benutzt und angepaßt ([Gro93]). Es stellt dem Benutzer sowohl Layoutfunktionen zur automatisierten graphischen Visualisierung des Graphen als auch Datenfilter zur Begrenzung der momentan auf dem Bildschirm dargestellten Information zur Verfügung. Der Benutzer steuert die Berechnung des Layouts durch Vorgabe einer Bewertungsfunktion, die die Qualität eines Layouts bewertet.

Beispiele für Komponenten der Bewertungsfunktion sind: gleichmäßige Verteilung der Knoten, Änderung des Layouts gegenüber einem Referenzlayout (Gewährleistung der topologischen Konsistenz in einer Sequenz von graphischen Darstellungen). Basierend auf dieser Vorgabe werden neue Layouts durch die implementierten Layoutalgorithmen generiert. Mit diesem Ansatz zeichnet sich GraphADL durch seine sehr gute Anpaßbarkeit an spezielle Problemstellungen (wie z.B. Logikplaneditor, Hypertext, Windowmanager) aus.

Abbildung 3 zeigt auf der linken Seite den Syntax–Graph–Teil des ES–Graphen dargestellt mit GraphADL. Rechts daneben ist das entsprechende C–Quellcode–Programm zu sehen.
Die vollständige Darstellung des entsprechenden Graphen auf einem Bildschirmfenster wäre zu unübersichtlich. Die Filterfunktionen haben automatisch nur die jeweils erste Hierarchiestufe dargestellt, d.h. Schleifenkörper wurden durch einen Knoten namens *body* dargestellt.
Mit Anwahl eines solchen Knotens kann die darunterliegende Hierarchiestufe sichtbar gemacht werden. Wenn allein schon die Anzahl der Knoten in der ersten Hierarchiestufe zur Unübersichtlichkeit führen würde, wird eine Auswahl von der Wurzel des Baumes her getroffen.
Es bestehen hier vielfältige Möglichkeiten, den Graphen zu analysieren bzw. zu durchsuchen: beispielsweise ist es auch möglich, die Korrespondenz eines Graphenknotens und der entsprechenden Instruktion im C–Quellcode darzustellen. Die im vorangegangenen Abschnitt aufgestellten Forderungen nach Debug–Möglichkeiten und benutzergestützter Partitionierung können ebenfalls erfüllt werden.

Durch einfache Anwahl eines Knotens im Syntaxgraphteil des ES–Graphen kann in den entsprechenden BSB gewechselt werden. Dazu zeigt Abbildung 4 einen Datenflußteil eines BSBs, auf dem vorher ein Scheduling ausgeführt wurde (die horizontalen gestrichelten Linien trennen dabei jeweils zwei Kontrollschritte). Diese Information kann dem Entwickler bei handunterstützter Partitionierung wichtige Entscheidungskriterien liefern.
Sind in einem DFG sehr komplexe Datenabhängigkeiten vorhanden, besteht auch hier die Möglichkeit, Teilmengen anzeigen zu lassen, die durch Optimierungsalgorithmen derart plaziert werden, daß die Anzahl der Überschneidungen der Graph–Kanten minimiert wird. In Abbildung 5 ist deshalb derselbe Ausschnitt aus dem Datenflußteil eines BSBs zu sehen, der auch in Abbildung 4 dargestellt wurde, allerdings ohne Benutzung des optimierenden Graphlayouts. Es ist deutlich zu sehen, daß die Anzahl der sich schneidenden Graphkanten erheblich verringert werden konnte, was wesentlich zur

Lesbarkeit des Graphen beiträgt.

Während der Einfluß des optimierenden Layouts in den BSBs des ES–Graphen also sehr wichtig ist, kommt der Syntaxgraphteil oft ohne optimierendes Layout d.h. mit einfacher Vorplazierung aus, wie Abbildung 3 zeigt. Dies liegt vor allem an der relativ einfachen weitgehend vorgegebenen Struktur dieses Graphteils.

Die wichtigsten Möglichkeiten, die sich durch Anwendung von GraphADL unter dem Co–Synthese System COSYMA bieten, sind:

- Debug–Möglichkeit in komplexen Benchmark Programmen (> 500 Zeilen C^x–Quell–Code) auf Graph–Basis wurde so überhaupt erst ermöglicht.
- Die Einarbeitungszeit in die ES–Graph–Syntax zum Zwecke der Erweiterung des COSYMA–Systems um neue Werkzeuge konnte erheblich reduziert werden.
- Handgesteuerte Partitionierung ist nun sehr einfach möglich und ergänzt die automatische Partitionierung.

5 Zusammenfassung

Die Erweiterung des COSYMA Systems durch die Visualisierung der Datenstrukturen hat den funktionalen Nutzen, daß nun neben der automatischen Partitionierung auch einfach handunterstützte Partitionierungen durch Markieren vorgenommen werden können.

Der praktische Nutzen ist vor allem in der einfachen Handhabbarkeit sowie der vereinfachten Fehlersuche in den teilweise sehr komplexen Graphteilen zu sehen.

Durch Anwendung der Erfahrungen im automatisierten Graphlayout ([LE92]) konnte darüberhinaus dafür gesorgt werden, daß die dargestellten Graphteile übersichtlich plaziert und Überschneidungen von Kanten möglichst verhindert werden.

Literatur

[AS93] P. Athanas, und H.F. Silverman, *Processor Reconfiguration Through Instruction–Set Metamorphosis,* IEEE Computer, S. 11–18, März 1993.

[BeHeEr93] Th. Benner, J. Henkel und R. Ernst, *Internal Representation of Embedded Hardware–/Software–Systems,* 2nd IFIP Int. Workshop on HW/SW Codesign, Codes/CASHE '93, Innsbruck, Mai 1993.

[BaRo92] E. Barros und W. Rosenstiel, *A Clustering Approach to Support Hardware/Software Partitioning,* Grassau, Germany, Umdruck vom IFIP Workshop on Hardware Software Codesign, Mai 1992.

[Gro93] H. Grosenick, *Erstellen einer graphischen Oberfläche für ein System zum Hardware/Software Co–Entwurf,* Diplomarbeit am Institut für Datenverarbeitungsanlagen der TU Braunschweig, Oktober 1993.

[ErHe92] R. Ernst und J. Henkel, *Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction,* International Workshop on Hardware-Software Co–Design, Estes Park, Colorado, Okt. 1992.

[ErHeBe93] R. Ernst, J. Henkel und Th. Benner, *Hardware/Software Co-Synthesis for Microcontrollers,* IEEE Design & Test of Computers, erscheint im Dez. 1993.

[GuMi92] R.K. Gupta und G.D. Micheli, *System-level Synthesis using Re-*

programmable Components,
EDAC'92, IEEE Comp. Soc. Press, S. 2–7, 1992.

[HeBeEr93] J. Henkel, Th. Benner und R. Ernst, *Hardware generation and partitioning effects in the COSYMA system,* IEEE/ACM International Workshop on Hardware–Software Codesign, Cambridge, Massachusetts, Okt. 1993.

[HBE+93] J. Henkel, Th. Benner, R. Ernst, W. Ye, N. Serafimov und G. Glawe, *COSYMA: A software–oriented Approach to Hardware/Software Co-Design,* The Journal of Computer and Software Engineering, 1993.

[HEH+93] A. Hoffmann, H.-J. Eikerling, W. Hardt, R. Genevriere, R. Camposano, *ISF, Eine Entwurfsdarstellung für High–Level–Synthese*, GI/ITG–Workshop für CAD–Umgebungen und Methoden des Entwurfs von Schaltkreisen und Systemen, Dresden, 1993.

[LaTho89] E. Dirkes Lagnese und D.E. Thomas, *Architectural Partitioning for System Level Design,* Proc. of 26th ACM/IEEE Design Automation Conference, IEEE Comp. Soc. Press, S. 62–67, 1989.

[LE92] P. Lüders und R. Ernst, *Verbesserung der Benutzeroberfläche von CAD–Systemen durch automatisches Bildschirmlayout*, Proc. of CAD '92, Springer Verlag, S. 77–93, 1992.

[YErBeHe93] W. Ye, R. Ernst, Th. Benner and J. Henkel, *Fast Timing Analysis for Hardware-Software Co-Synthesis,* Cambridge, Massachusetts, Proc. of Int. Conf. on Computer Design 1993, IEEE Comp. Soc. Press, S. 452–457, Okt. 1993.

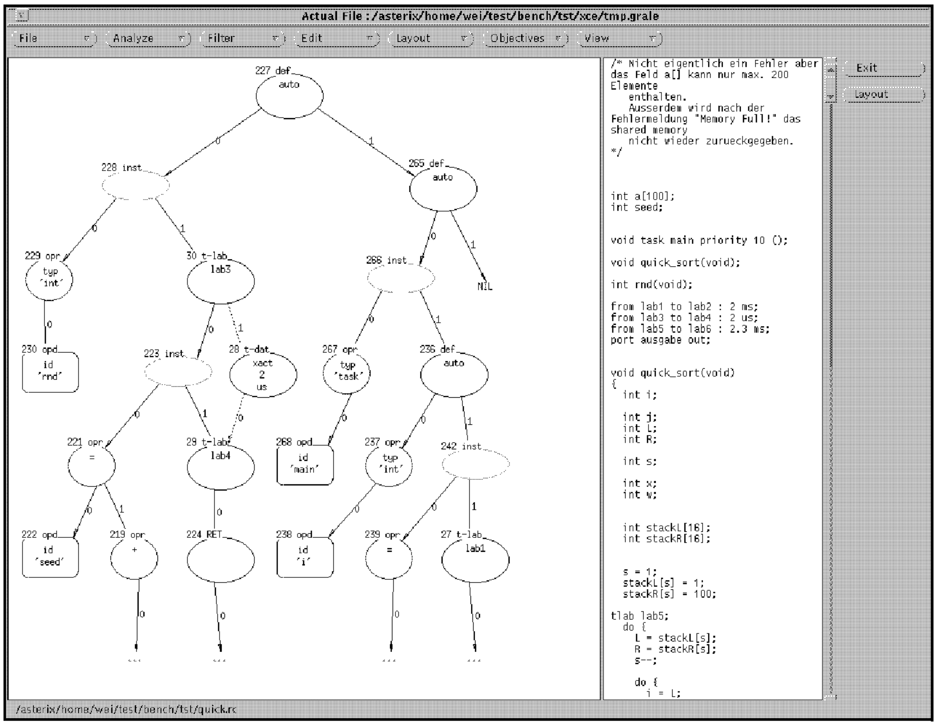

Abbildung 3: Syntaxgraphdarstellung ohne optimierenden Plazierungsalgorithmus

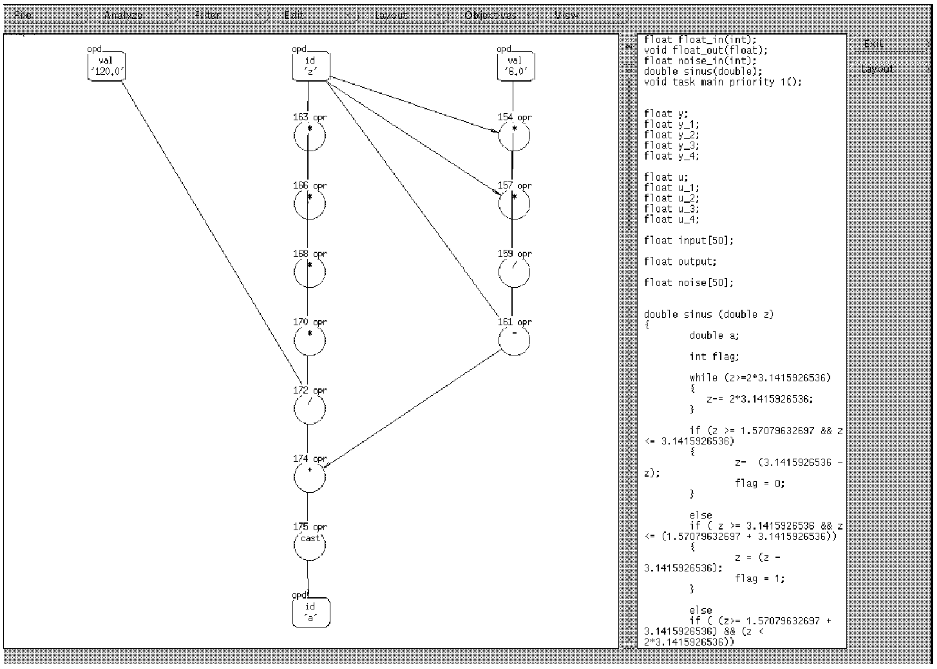

Abbildung 4: Datenflußgraphdarstellung mit optimierenden Plazierungsalgorithmus

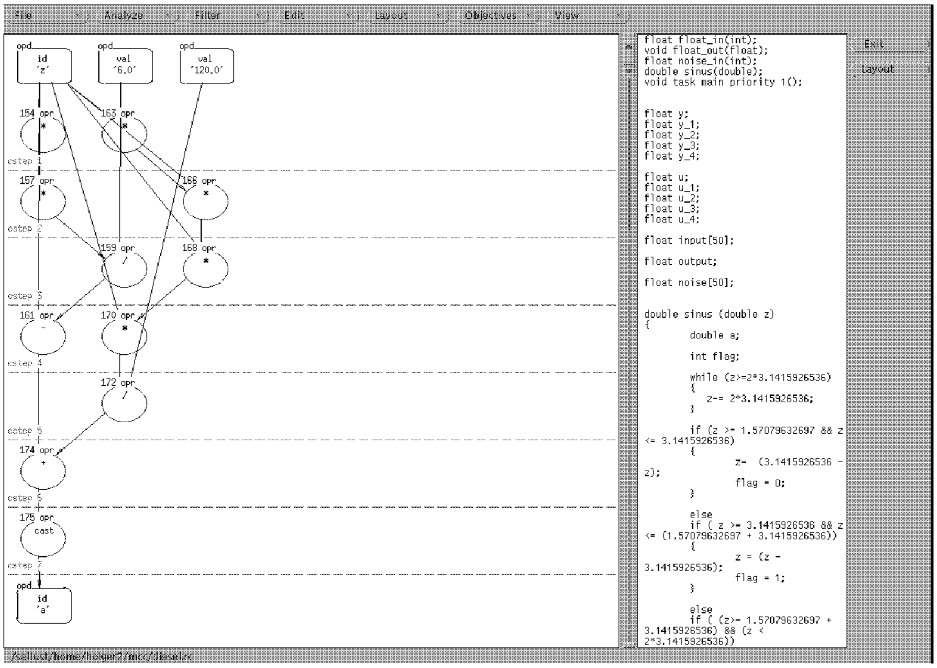

Abbildung 5: Datenflußgraphdarstellung mit opt. Plazierungsalgorithmus