Diss. ETH Nr. 9326

Design of a Variable Grain Dataflow Machine and its Relation to a New Approach for System Specification

ABHANDLUNG
zur Erlangung des Titels
DOKTOR DER TECHNISCHEN WISSENSCHAFTEN
der
EIDGENÖSSISCHEN TECHNISCHEN HOCHSCHULE
ZÜRICH

vorgelegt von
Jürg A. Wyttenbach
Dipl. Informatik Ing.ETH
geboren am 18 Oktober 1958
von Gerzensee

angenommen auf Antrag von
Prof. Dr. A. Kündig, Referent
Prof. Dr. E. Engeler, Korreferent

1991

Kurzfassung:

Die vorliegende Arbeit entstand als Zusammenfassung des Wissens, das beim Entwurf eines parallelen Datenflussrechners erworben wurde. Der architekturspezifische Teil der Arbeit befasst sich mit den innovativen Aspekten einer grobkörnigen Datenflussarchitektur. Der vollständige Entwurfsansatz der Codeblock Metamaschine (abstrakte Maschine) wird vorgestellt und als effiziente Lösung der Parallelisierung von Berechnungen eingeführt. Die vorgestellte Architektur wurde bereits durch einen Simulator (realisiert in Modula2) ausgetestet und wird nun in einer Konkretisierungsphase auf eine möglichst effiziente Hardware - Implementierung vorbereitet. Die Beschreibung der Simulationsaspekte etc. ist nicht Gegenstand dieser Arbeit; diese werden in einer weiteren Dissertation von O. Maquelin beschrieben werden.

Ein wichtiger Punkt zum Verständnis paralleler Maschinen ist ihre Programmierung. Es hat sich gezeigt, dass eine parallele Ausführung von Berechnungen notwendigerweise auf partiellen Ordungen über Termen beruht. Zwei Terme, deren transitive Hüllen unabhängig sind, können parallel berechnet werden. Solche Hüllen werden am einfachsten durch Datenflussgraphen (gerichtete, azyklische Graphen) dargestellt, die wiederum elegant von funktionalen (seiteneffektfreien Sprachen) abgeleitet werden können. Somit wäre das zweite Stichwort dieser Arbeit gefallen, welches sich auf den Design einer funktionalen, single Assignment Sprache bezieht. Der Entwurf einer "neuen" Sprache hat sich aus mehreren Gründen aufgedrängt: 1) Als Benchmarking Tool für unsere parallele Maschine wurde aus naheliegenden Gründen eine funktionale (erster Ordung) Sprache gewählt, deren Ziel es war eine möglichst effiziente Abbildung von Algorithmen auf eine verteilte Maschine zu garantieren. 2) Parallele Programme stellen höhere Anforderungen bezüglich Korrektheit. Die meisten gängigen Sprachen, insbesondere Modula2, besitzten nicht einmal die minimalsten Spezifikationshilfen für den Entwurf konsistenter Programme. So wird im Falle von Modula2 gänzlich darauf verzichtet über haupt auf eine abstrakte Maschine Bezug zu nehmen. 3) Die Definition grosser verteilter Systeme braucht immer mächtigere und auch besser definierte Hilfsmittel (Tools). Das Verständnis aller Aspekte einer entsprechenden Umgebung kann nur erworben werden, in dem man auf allen Ebenen aktiv Erfahrungen sammelt. Als Konsequenz basiert die Definition von MFL, unserer funktinalen Kernsprache, auf einem hierarchischen Model von Algebren, die es erlauben (Daten-) Typen erstmals konsistent und vollständig zu spezifizieren.

Der dritte Teil der Arbeit enstand aus der Analyse und Detailspezifikation der ersten beiden Teile quasi von selbst, als eine innere Notwendigkeit, die Natur von abstrakten System besser zu verstehen. Immer wieder wurde der Autor beim Entwurf der Maschine oder der Sprache das Opfer von unzulänglichen Spezifikationshilfsmitteln. Sei dies eben EBNF oder attributierte Grammatiken (ergänzt durch Prädikaten-Logik oder Register - Register - Transfer Spezifikationen von Hardware), immer wieder stand das gleiche Problem zu Diskussion: Die gewählten Formalismen beschreiben **einen** Aspekt zwar vollständig, sind aber als Basis für andere, weitergehende Beschreibungen oft völlig ungeeignet. Als Ausweg aus dem Dilemma wurde versucht einen Weg des integralen Systementwurfs, - kurz: Die Abstraktion (das Weglassen) von Information, sowie deren Gegenteil, die Spezifikation - das Beschreiben (Hinzufügen) von Information, - zu analysieren. Die dabei erworbenen Erkenntnisse

wurden beim Entwurf einer vollständigen Beschreibung von MFL erstmals eingesetzt. Es ist nicht zu verkennen, dass das parallele Arbeiten an allen Ebenen des Architekturentwurfes diese Arbeit stark befruchtet, ja erst ermöglicht hat. Als Kosequenz aus diesem dritten Teil der Arbeit kann schon heute gesagt werden, dass Programmiersprachen wie wir sie heute grösstenteils noch verstehen, über kurz oder lang verschwinden werden. An ihre Stelle werden hierarchische Designsysteme treten die es erlauben Abstraktionen konsistenzerhaltend zu erweitern. Doch das ist Zukunftsmusik die erst verstanden werden muss.

Abstract:

This thesis originated in the wish to compile the know-how acquired in designing and developing a codeblock dataflow computer. In a first section the innovative aspects of a codeblock dataflow machine are discussed. The complete abstract model of a codeblock dataflow metamachine is presented and the efficiency of the solution is discussed. The model under discussion has been implemented as a simulation/emulation model using MODULA2. Such a software realization allows to fine tune the design in respect to a final hardware realization. The discussion of the simulator is not part of this theses.

Understanding parallel machine also requires deep knowledge in how they will be programmed. In general parallel evaluation is based on the partial order of terms. Two terms with a non intersecting transitive closure may be evaluated in parallel. Transitive closures are most easily represented by dataflow graphs (directed acyclic graphs), which may easily be derived from functional (side effect free) languages. This leads directly to the second part of this thesis, discussing the design and definition of a "new" functional single assignment language.

The design of a "new" language was a prerequisite for further work: 1) A benchmarking tool for our metamachine was needed and it was obvious to choose a functional (first order) language because we were interested mainly in numerical computations. A functional language is also an ideal base for coding algorithms that may easily be distributed on a parallel machine. 2) Because parallel algorithms are even more demanding in respect to their correctness than their sequential counterparts (typically expressed in Modula2 which has no relation defined to an abstract machine) we tried to define the language as tight as possible using algebraic formalisms. 3) Defining, specifying large systems, becomes more and more demanding for powerful tools. Only if all aspects of the system design - life-cycle are fully under control , the research may provide us with experience and finally with answers to actual problems. As a consequence the whole systems will finally be defined using algebraic approaches, especially for MFL (the functional kernel language) we already outlined a promising way to reach a specification that is very concise and short. Finally some innovative solutions to compiling problems of MFL source are presented and the future development of our language / specification environment is outlined.

A third part which heavily influenced the design of our environment has originally be integrated in this work but it became apparent that it was even more fundamental than we expected and thus will be treated elsewhere. Nevertheless the ideas gained with studying new methods for abstract system design were employed to develop a consistent hierarchy for algebraically defining MFL.