Parallelität auf Befehlsebene

ein Arbeitsprogramm

Gerhard Goos
Fakultät für Informatik
KIT Karlsruhe

26. Januar 2013

Inhaltsverzeichnis

1 Übersicht

Gegeben sei ein Rechensystem mit p Rechnerkernen und ein sequentielles Programm π in optimierter SSA-Form, z. B. in Firm. Wir unterteilen Grundblöcke in noch kleinere Einheiten, die wir parallel auf den gegebenen Rechnerkernen ausführen wollen, um die gesamte Ausführungszeit zu verkürzen.

2 Erster Schritt

Wir folgen den Ideen von Kiran et al., (Kiran et al., 2011a,b, 2012a,b,c) und unterteilen jeden Grundblock g in kleinere Einheiten, die wir hier *Elementarblöcke* nennen. Zwei Operationen $op1, op2 \in g$ gehören zum gleichen Elementarblock, wenn $op2$ datenabhängig von $op1$ ist. ϕ – Funktionen und Konstante können zu mehreren Elementarblöcken gehören; ansonsten sind Elementarblöcke disjunkt.

Das Ergebnis ist eine Art Petri-Netz mit einer Eingangs- und einer Ausgangsecke. Ecken sind die Elementarblöcke. Es gibt zwei Arten von Kanten, nämlich Ablaufkanten und Datenabhängigkeitskanten.

Eine Ecke kann schalten, wenn alle Vorgänger, von denen Daten in der Ecke abhängen, und genau eine eingehende Ablaufkante belegt sind. Bei Datenabhängigkeit bleiben die Marken der Vorgänger erhalten, bis der Vorgängerblock erneut betreten wird (bei Schleifen). Für Ablaufkanten wird die Marke im Vorgängerblock gelöscht. Am Ende eines Elementarblocks werden Marken für alle Datenabhangigkeits- und Ablaufkanten gesetzt.

3 Zweiter Schritt

Auf der Grundlage einer vorläufigen Codeerzeugung betreiben wir lokale Registerzuordnung für die Elementarblöcke. Wichtig ist allein, welche Werte (einschließlich des Ergebnisses initialer ϕ – Funktionen) in Registern gehalten werden und wieviele Register hierdurch belegt werden. Ferner ermitteln wir, welche Werte ausgelagert werden sollen.

4 Dritter Schritt

Auf der Grundlage der vorangehenden Schritte schätzen wir die Ausführungszeit der Elementarblöcke nach oben ab. Solange wir nichts über die Belegung der *cache*-Speicher wissen, müssen wir dabei davon ausgehen, daß Speicherzugriffe tatsächlich Hauptspeicherzugriffe ohne *cache*-Benutzung sind.

5 Vierter Schritt

Jetzt können wir die Elementarblöcke zur Ausführung auf die vorhandenen p Rechnerkerne verteilen.

Kriterien dabei sind:

- Elementarblöcke, zwischen denen eine hohe Anzahl von Datenabhängigkeiten besteht, auf den gleichen Rechnerkern;
- Minimierung der expliziten Synchronisierung zwischen aufeinanderfolgenden Elementarblöcken auf verschiedenen Rechnerkernen unter Nutzung der Kenntnisse über Ausführungszeiten;

Diese Kriterien sind nicht notwendig konsistent.

Kiran et al. machen Vorschläge für dieses *scheduling*. Eine andere Möglichkeit ist die Verwendung des LogP-Modells. Löwe (1996) beweist in seiner Dissertation, daß man weniger als das Doppelte der absolut optimalen Ausführungszeit erreichen kann.

6 Fünfter Schritt

Unter Kenntnis des *scheduling* und einer (endgültigen?) Codeerzeugung können wir jetzt eine globale Registerzuteilung für die einzelnen Prozessoren machen und dann Aussagen über die Belegung der *caches*. Beides zusammen führt zu neuen Abschätzungen der Ausführungszeiten. Diese Kenntnis läßt sich zur Reduzierung der expliziten Synchronisierung nutzen. Man kann aber auch iterativ vorgehen und mit den jetzt vorhandenen Kenntnissen Schritt 4 und 5 wiederholen.

Literatur

Kiran, D., Gurunarayanan, S. und J.P., M. (2012a): Compiler Driven Inter Block Parallelism for Multicore Processors. In *Wireless Networks and Computational Intelligence*, herausgegeben von Venugopal,

K. R. und Patnaik, L. M., Bd. 292 von *Communications in Computer and Information Science*, S. 426–435. Springer Berlin Heidelberg. ISBN 978-3-642-31686-9. 10.1007/978-3-642-31686-9_50.

Kiran, D., Gurunarayanan, S. und Misra, J. (2011a): Taming Compiler to Work with Multicore Processors. In *Process Automation, Control and Computing (PACC), 2011 International Conference on*, S. 1 –6.

Kiran, D., Radheshyam, B., Gurunarayanan, S. und Misra, J. (2011b): Compiler Assisted Dynamic Scheduling for Multicore Processors. In *Process Automation, Control and Computing (PACC), 2011 International Conference on*, S. 1 –6.

Kiran, D. C., Gurunarayanan, S., Khaliq, F. und Nawal, A. (2012b): Compiler Efficient and Power Aware Instruction Level Parallelism for Multicore Architecture. In *Eco-friendly Computing and Communication Systems*, herausgegeben von Mathew, J., Patra, P., Pradhan, D. K. und Kuttyamma, A. J., Bd. 305 von *Communications in Computer and Information Science*, S. 9–17. Springer Berlin Heidelberg. ISBN 978-3-642-32112-2. 10.1007/978-3-642-32112-2_2.

Kiran, D. C., Gurunarayanan, S., P., M. J. und Khaliq, F. (2012c): An Efficient Method to compute Static Single Assignment Form for Multicore Architecture. In *1st. Int Conference on recent Advances in Information Technology TRAIT 2012*.

Löwe, W. (1996): *Optimierung paralleler Programme*. Berichte aus der Informatik. Shaker. ISBN 9783826530685.