Heterogene Paralelle Rechensysteme

Gedächtnisprotokoll vom 27. März 2018

Note: 1,0

1 Superskalare Pipelines

Aufgabe: Diagramm der superskalaren Pipeline aus der Vorlesung aufzeichnen. Die Buffer und die Stufen bennenen.

Frage: Welche Konflikte gibt es bei einer Pipelinearchitektur?
Antwort: Datenkonflikte, Namenskonflikte, Strukturkonflikte und Steuerkonflikte.

Frage: Wie lassen diese sich lösen? Welche Stufen der Pipeline sind dafür verantwortlich?
Antwort: Die Steuerkonflikte werden (hinreichend gut, aber nicht immer korrekt) von der Sprungvorhersage aufgelöst. Diese befindet sich auf Höhe der Instruction Fetck Einheit. (Diagramm habe ich dann um diese erweitert). Namenskonflikte löst die Instruction Decode Einheit auf, indem sie von außen sichtbare Register auf interne Registernamen abbildet. Strukturkonflikte löst die Dispatch Einheit, die erst einen Befehl in die Issue Buffer legt, wenn dort Platz ist (also die Execution Einheit frei ist). Die Datenabhängigkeiten lassen sich ebenfalls in der Dispatch Einheit auflösen (hiermit bin ich mir nicht sicher, aber es wurde auch nichts weiter dazu angemerkt). Dann habe ich noch Tomasolu erwähnt. Es wurde nicht weiter darauf eingegangen.

2 Parallelisierungsprozess

Frage: Wir hatten die Fallstudie OCEAN betrachtet. Wie war dort das Vorgehen, um aus einem sequentiellen Programm ein paralelles zu machen?
Antwort: Ich habe die vier Schritte der Dekomposition, Zuweisung, Festlegung und der Abbildung aufgezeichnet und erklärt.

Frage: Wenn ich nun das mit einem paralellen Programmieransatz diesen Ablauf realisiere, wo greift dieser ein?
Antwort: Ehh was? Ich meinte dass zum Beispiel OpenMP bei der Aufteilung

eingreift. Das war aber falsch, denn das muss der Programmierer noch selber festlegen. Er wollte hören, dass ein paraleller Programmmieansatz bei der Festlegung greift, da dort die Kommunikation letzten Endes festgelegt wird. Dies macht nicht der Programmierer bei OpenMP. Sehr obskure Frage und Antort. Nicht aus der Ruhe bringen lassen....

Frage: Welche Möglichkeiten habe ich denn, ein Programm zu paralellisieren?
Antwort: Es gibt zum einen die Funktionsdekomposition. Zum Anderen die Domänendekomposition

Frage: Welches Verfahren ist bei OCEAN geeignet?
Antwort: Da lediglich Gleichungssysteme zu lösen sind, und diese Einzeiler sind, kann man schlecht Funktionsdekomposition betreiben. Daher ist das aufteilen der Daten sinnvoller. Also Domänendekomposition.

Frage: Wie wurde dies umgesetzt?
Antwort: Da gab es verschiedene Ansätze. Einmal das Jakobi-Verfahren, das Gauss-Seidel-Verfahren und die Methode, die Datenabhängigkeiten ignoriert. Ich habe das die „asynchrone Methode"genannt (Vergleiche Folien WS17/18, Satz 3, Folie 46 zweiter Bulletpoint), was aber für eine fehlerhafte Bennenung gehalten wurde.

Frage: Welches von denen sorgt für eine ausbalancierte Aufgabenverteilung?
Antwort: Jakobi und Gauss-Seidel fallen da raus, da die Iterationen verschieden viele Daten erhalten (die Diagonalen sind ja zum Beispiel verschieden lang). Ich meinte aber, dass man das mit #pragma omp paralell for schedule(dynamic) beheben konnte. Der Wortlauf darauf war (wortwörtlich): „OpenMP ist mir jetzt mal egal!". Die richtige antwort war dann die asynchrone Methode, da dort die Daten gleichmäßig in Zeilen eingeteilt werden.

3 Kommunikationsmodelle

Frage: Wie sieht die Kommunikation von Threads bei OpenMP aus?
Antwort: Implizit durch geteilte Variablen im gemeinsamen Speicher.

Frage: Und wenn kein gemeinsamer Speicher existiert?
Antwort: Dann braucht man so etwas wie ein Message Passing Interface.

Frage: Welche grundlegenden Entscheidungen muss ich bei der Implementierung dieser Nachrichten fällen?
Antwort: Da gibt es die synchrone- und asynchrone Variante, sowie die blockierende- und nichblockierende Eigenschaft. Synchron bedeutet, dass beide Prozesse aufeinander warten, bis der Nachrichtenaustausch stattgefunden hat. Bei der asynchronen Variante hingegen kann der Prozess weiter rechnen und ein Sende- bzw.

Empfangsprozess kümmert sich um die Kommunikation. Wenn dieser blockierend ist, erhält der rechnende Prozess erst dann wieder die Kontrolle wieder zurück, sobald die Daten in dem entsprechenden Sendebuffer gelegt oder aus dem Empfangsbuffer in den lokalen Bereich kopiert wurden. Ist dieser nichtblockierend, kann der Prozess sofort weiterrechnen und muss mit einer Probefunktion gucken, ob inzwischen der Kommunikationsprozess ausgeführt wurde.

Frage: Die synchrone Variante ist deutlich langsamer. Wieso?
Antwort: Weil damit gleichzeitig die Prozesse synchronisiert werden. Diese Variante vereinigt das Prinzip der Kommunikation und der Synchronisation.

4 Heterogene Strukturen

Frage: Was versteht man unter einer heterogenen Rechnerstruktur?
Antwort: Ein Verbund aus mehreren, nicht gleichförmigen Recheneinheiten. Wie zum Beispiel der Verbund einer GPU mit einer CPU.

Frage: Wie lassen diese sich verbinden?
Antwort: Einerseits Offchip. Über einen PCIe Bus über die Nordbrücke. Ich habe dazu ein kleines Bildchen gemalt und mich prompt verquaselt und gesagt, dass der Front Side Bus bei Intel QPI und bei AMD HT 3.0 heißt. Aber in Wirklichkeit ersetzen diese beiden Technologien den veralteten FSB.

Frage: Was sind dabei die Vor- und Nachteile?
Antwort: Offchip ist flexibel, dafür sind aber nötige Datentransfers zeitlich teuer. Onchip hingegen sind diese gar nicht da, bei gemeinsamen Speicher. Dafür sind diese absolut nicht flexibel.

Frage: Welches Programmiermodell nutzt man, um diese gemeinsam rechenen zu lassen?
Antwort: OpenCL. Oder, wie hier am Lehrstuhl geforscht, mit HAL.

Frage: Wie sieht der Ablauf einer OpenCL Programmierung aus?
Antwort: Zuerst hat man seinen Quellcode und compiliert ihn mit einer Standardcompiler. Dieser erzeugt eine Anwendung mit OpenCL Kernel. Diese geht durch die OpenCL API und erzeugt dann, je nach Bibliothek einen Hardwarspezifischen Compiler, der dann den Kernel für die Hardware kompiliert. Ich habe das Schaubild dazu mit den vier Schritten gemalt.

Frage: Wo in diesem Prozess wäre HAL verankert?
Antwort: Bevor es zu den Bibliotheken geht. Dazwischen sitzt eine Datenbank (und noch mehr), die für ein gegebenes Problem und dessen Eingabegröße erkennt, welche Hardware gerade am schnellsten diese Aufgabe lösen kann. Es wird ein Tradeoff zwischen Datentransferzeit und danach möglichem Speedup gefunden. Das passiert zur Laufzeit und für den Nutzer transparent.