PARASITIC COMPUTING
Auswertungen & Systemtest

Version 1.0

Jürg Reusser
Luzian Scherrer

3. Januar 2003

Zusammenfassung

Dieses Dokument zeigt die Resultate und Erkenntnisse, welche sich aus der im Rahmen der Diplomarbeit „Parasitic Computing" entwickelten Software ergeben haben. Im Folgenden werden verschiedene Parameterisierungen und Ausführungsarten der Software auf ihre Effizienz getestet und verglichen. Den zweiten Teil dieses Dokumentes bildet ein Bericht über das Vorgehen und die Resultate des Systemtests, mittels welchem die Korrektheit der Programme sichergestellt wurde.

Inhaltsverzeichnis

1 Auswertungen

1.1 Parallelisierung der Addition

Die der virtuellen Maschine zugrundeliegende Operation ist die Ganzzahladdition auf Bitebene. In ihrer atomaren Form geschieht diese, wie gezeigt wurde, sequentiell für jede Bitstelle der zu addierenden Zahlen. Es ist aber auch denkbar, in einem Sendeimpuls[1] die Lösungen von breiteren als 1-Bit Additionen zu prüfen und somit eine Parallelisierung der Grundoperation zu erreichen.

1.1.1 Anzahl benötigter Netzwerkpakete

Die Anzahl betötigter Netzwerkpakete der Addition entspricht

$$AnzahlPakete = \frac{RB}{x} \times 2^{(x+1)}$$

wobei *RB* die Registerbreite der virtuellen Maschine in Bits und *x* die Parallelisierungskonstante ist. Letztere definiert, wieviele Bitstellen pro Sendeimpuls berechnet werden sollen. Für die Parallelisierungskonstante gibt es zwei Einschränkungen. Erstens muss sie ein ganzer Teiler der Registerbreite sein, denn nur so lässt sich das ganze Register gleichmässig aufteilen, zweitens darf sie nicht höher als 4 sein. Der nächstmögliche Wert nach 4 wäre 8, dies geht aber bereits nicht mehr, da sich – die Netzwerkpakete sind 16-Bit breit – Operatoren und Checksummen überschneiden würden.

Somit sind, von einer 8-Bit Registermaschine ausgehend, nur die ersten drei in Tabelle 1 gezeigten Kombinationen möglich. Die restlichen Werte dienen er Anschaulichkeit.

RB	*x*	Anzahl Pakete
8	1	32
8	2	32
8	4	64
8	8	512
8	16	65’536
8	32	2’147’483’648

Tabelle 1: Anzahl Pakete abhängig von der Parallelisierung

1.1.2 Laufzeit und Prozessorzyklen

Die folgende Abbildung 1 zeigt die unterschiedlichen Laufzeitresultate desselben Programmes, ausgeführt in den drei verschiedenen Parallelisierungsmodi. Das Testprogramm ist eine einfache Multiplikation, wie sie in den 4IA-Codebeispielen zu finden ist. Die abgebildeten Ergebnisse sind die aus 15 sukzessiven Durchläufen ermittelten Durchschnittswerte im parasitären Modus. Als Hardware für diesen Test diente eine Sun Ultra Sparc 1 mit einer Taktrate von 167 MHz.

[1] Nach dem beschriebenen Prinzip der Kandidatlösungsprüfung benötigt die Berechnung eines Teilresultates mehrere Netzwerkpakete. Diese Ansammlung an gesendeten Paketen pro Teilresultat bezeichnen wir als Sendeimpuls.

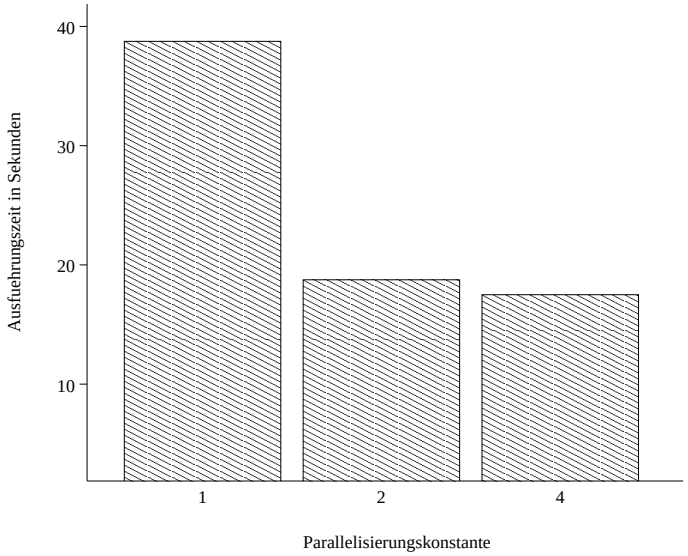

Abbildung 1: Unterschiedliche Laufzeiten je nach Parallelisierungsart

Es zeigt sich, dass die höchstmögliche Parallelisierung das beste Laufzeitverhalten ergibt. Demgegenüber muss jedoch auch der Aufwand an Prozessorzyklen der ausführenden Maschine betrachtet werden. Es entspricht dem eigentlichen Sinn des parasitären Rechenwerkes, diesen Wert so klein wie möglich zu halten. Die Abbildung 2 zeigt die benötigten Prozessorzyklen pro Parallelisierungsmodus für das gleiche Testprogramm, ermittelt jeweils wieder als Durchnittswerte von 15 Durchläufen.

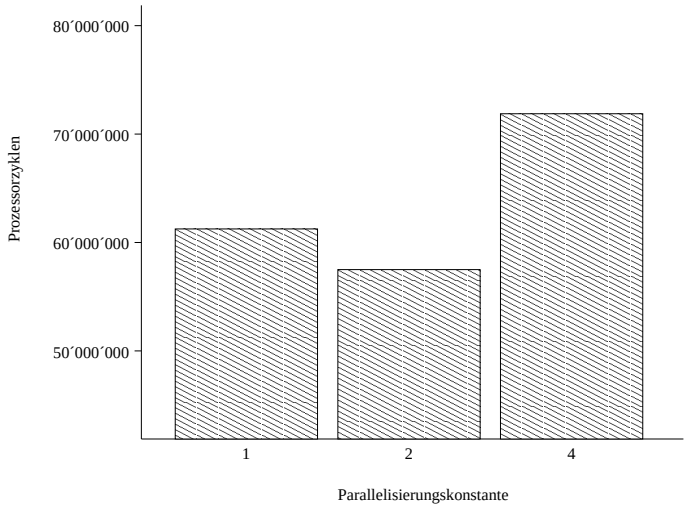

Abbildung 2: Benötigte Prozessorzyklen je nach Parallelisierungsart

Aus den beiden Grafiken wird ersichtlich, dass die Variante mit einer Parallelisierungskonstante von 2 die optimalsten Resultate liefert. Dies entspricht auch den Erwartungen

aus obengenannter Formel, denn $x = 2$ ist der Optimalfall aus den Anforderungen:

- Geringe Anzahl an ICMP Paketen
  Die Werte $x = 1$ und $x = 2$ benötigen beide die gleiche, kleinstmögliche Anzahl an ICMP Paketen. Das Bilden dieser Pakete ist betreffend der benötigten Prozessorzyklen eindeutig die „teuerste" Operation.

- Bestmögliche Parallelisierung
  Eine höhere Parallelisierung hat zur Folge, dass weniger oft auf die Timeouts eines Sendeimpulses gewartet werden muss. Deshalb gilt – wie aus Abbildung 1 ersichtlich – der Grundsatz: je höher die Parallelisierung, desto kleiner die Ausführungszeit.

Der Defaultwert für die Parallelisierungskonstante ist somit bei Applikationsstart immer 2.

1.2 Gegenüberstellende Messungen

In den nachfolgenden Abschnitten sind zwei gegenüberstellende Messungen und deren Resultate als Verhältnisse aufgezeigt. Es ist zu beachten, dass es sich bei den aufgeführten Werten um statistische Ergebnisse von Grössen handelt, welche in der Praxis variieren können. Das Hauptziel dieser Messungen ist es, den ungefähren Rahmen der Grössenordnungen abzustecken.

1.2.1 Reelle vs. virtuelle Prozessorzyklen

Jede Iteration der in [RS02a] gezeigten Schlaufe der virtuellen Maschine entspricht zwei virtuellen Prozessorzyklen. Ein Zyklus wird dabei für die jeweils ausgeführte Instruktion, ein weiterer für die Inkrementierung des Instruktionszeiger-Registers gezählt.

Das Verhältnis aus den benötigen Prozessorzyklen der virtuellen und denjenigen der diese ausführenden Maschine kann nicht generell angegeben werden, da es erstens, wie vorgängig erläutert, von der Parallelisierungskonstante und zweitens vom aktuell berechneten Problem, sprich dessen Instruktionen, abhängt. Um jedoch eine ungefähre Grössenordnung als Anhaltspunkt zu nennen, sei hier auf das der Distribution beiliegende Programm `bubblesort.xia` verwiesen, welches im Simulationsmodus mit einem ungefähren Verhältnnis von

1 : 7'000

virtuellen zu reellen Prozessorzyklen ausgeführt werden kann. Dieses Resultat hat sich auf einem Pentium III mit einer Taktrate von 1 GHz und dem Betriebssystem Linux 2.4.7 ergeben. Im parasitären Modus mit gleichem Programm und gleicher Konfiguration beläuft sich das Verhältnis auf ungefähr

1 : 600'000

virtuelle zu reellen Prozessorzyklen (bei einer Ausführungszeit von etwas mehr als 13 Minuten, 8'760'289 gesendeten und 1'095'036 empfangenen ICMP Paketen).

In der selben Umgebung erzielte das ebenfalls in der Distribution enthaltene Programm `fibonacci.xia` ein Verhältnis von

1 : 9’500

virtuellen zu reellen Prozessorzyklen, ebenfalls im Simulationsmodus. Dieses Programm lässt sich im parasitären Modus nicht in sinnvoller Zeit ausgeführen, weshalb hier kein demenstprechender Prozessorzyklen-Vergleich angegeben werden kann.

1.2.2 Häufigkeit falscher Checksummen

Um eine Ahnung davon zu bekommen, wie oft Pakete mit falschen Checksummen in einem durchschnittlich belasteten Netzwerk auftreten, ist für diese Arbeit eine Testmessung vorgenommen worden. Dabei wurde an einem Cisco Catalyst Switch mit 9 angeschlossenen Hosts ein SPAN[2] konfiguriert und daran mit der Software *Etherreal*[3] wärend der Dauer von einer Woche alle Pakete mit falschen Checksummen gefiltert. Dabei wurden von insgesammt 22’912’281 gemessenen Paketen gerade mal 12 mit nicht korrekten Checksummen vom Filter aufgefangen. Dies entspricht einem ungefähren Verhältnis von

1 : 2’000’000

Paketen mit falschen zu Paketen mit korrekten Checksummen. Das Verhältnis schient gering, allerdings benötigt beispielsweise die Ausführung des der Distribution beiliegenden Programmes `fibonacci.xia` zur Berechnung der ersten 20 Fibonacci-Zahlen bereits 201’752’880 ICMP Pakete.

Pakete mit falschen Checksummen können die Berechnung der virtuellen Maschine allerdings nur dann zu Fall bringen, wenn auf einen Sendeimpuls eines (und genau eines) der falschen Pakete als korrekt und alle anderen Pakete (inklusive das einzelne korrekte) als falsch behandelt werden. Es müssten also die Checksummen von exakt zwei Paketen verändert werden, nämliche diejenige des korrekten Paketes als falsch und diejenige eines falschen Paketes als korrekt. Ein solcher Fall ist kaum denkbar (siehe Abschnitt ,,Algorithmus: parasitäre Verteilung” in [RS02b]).

[2] Switched Port Analyzer
[3] siehe `http://www.ethereal.com/`

2 Systemtest

2.1 Generelles Vorgehen

Wärend der gesamten Entwicklungsphase wurde der Fehlerprüfung stets höchste Gewichtung zugemessen, denn die Korrektheit ist bei Softwareentwicklungen das wichtigste und schlussendlich über Erfolg oder Misserfolg entscheidende Kriterium. Da diese Diplomarbeit im Team von zwei Personen entwickelt wurde, konnten wir durch gegenseitiges Audit der jeweils neu erstellten Codeabschnitte potentiell eingeführte Fehler immer kurzerhand eliminieren. Ausserdem haben wir den Testprozess der „Ausführungskorrektheit" weitgehend automatisiert, wie im nachfolgenden Unterkapitel beschrieben wird.

Vor Abschluss der Implementationsphase wurde nochmals ein komplettes, die gesamten Softwarepakete umfassendes Code-Audit durchgeführt, wobei alle erstellten Programme Zeile für Zeile verifiziert wurden.

Mit der endgültigen Fassung der Software ist es uns nicht mehr gelungen – sei es im parasitären- oder im Simulationsmodus – falsche Resultate bei der Programmausführung zu erhalten.

2.2 Das `pshell_test.pl` Utility

Die in der Distribution enthaltenen 4IA und XIA Programmbeispiele umfassen die kompletten Möglichkeiten der beiden Sprachen. Wenn alle diese Programme korrekt ausgeführt werden können, so gilt der Systemtest als bestanden. Um diesen Prozess wärend der Entwicklungphase – nach jeder Änderung der Software musste deren Korrektheit weiterhin sichergestellt werden können – zu automatisieren, haben wir das Utility `pshell_test.pl` geschrieben. Es ist eine einfache, in PERL[4] implementierte State-Event Maschine, welche für jedes enthaltene Programmbeispiel eine `pshell` Sitzung durchführt und die erwarteten Resultate überprüft. Die entsprechenden Ergebnisse des Tests werden dabei auf der standard Ausgabe angezeigt.

Das `pshell_test.pl` Utility ist auf der beiliegenden CD-ROM, beziehungsweise unter `http://parasit.org/code/`, verfügbar.

[4] PERL: „Practical Extraction and Reporting Language", siehe `http://www.perl.com/`.

Literatur

[RS02a] Juerg Reusser and Luzian Scherrer. Parasitic computing: Realisierungskonzept. `http://parasit.org/documentation`, Oktober 2002.

[RS02b] Juerg Reusser and Luzian Scherrer. Parasitic computing: Systemdesign. `http://parasit.org/documentation`, Dezember 2002.