Hardwarebeschleunigung von paralleler Ablaufplanung für Multicoresysteme
Zwischenbericht zum 2. Jahr

Nina Engelhardt

19. Juni 2013

Die Supercomputer von Gestern sind die eingebetteten Systeme von Morgen. Massive Anzahlen von Prozessorkernen, wie man sie einst nur in Rechenzentren fand, halten bereits Einzug in Autos. Damit weitet sich das altbekannte Problem aus, dass es viel schwerer ist, Programme zu schreiben, die viele Kerne gleichzeitig effizient benutzen können, als Programme für nur einen Kern zu schreiben.

Ein vielversprechender Lösungsansatz sind dömänenspezifische parallele Programmiersprachen [7]. Die Programmierung gelingt schneller und fehlerfreier, wenn die Konzepte der Programmiersprache denen der Anwendung angepasst sind. Allerdings haben spezifischere Programmiersprachen eine kleinere Nutzerbasis, sodass sich die Entwicklungskosten für die Sprache und die dazugehörigen Tools schnell nicht mehr rentieren. Das Virtualized Master-Slave Framework[5], kurz VMS, das Subjekt meines Dissertationsvorhabens ist, erleichtert die Entwicklung von Runtimes für parallele Programmiermodelle, indem es grundlegende Funktionalitäten wie Synchronisation übernimmt. Um ein bestimmtes Modell zu unterstützen, muss der Entwickler nur noch einige Funktionen in Form eines *Plugins* hinzufügen. Diese Pluginfunktionen sind einfach zu schreiben, da sie die üblichen strengen Auflagen wie Thread-safety oder Reentrancy nicht erfüllen müssen.

Damit diese Strategie Erfolg hat, sollten die resultierenden Runtimes ähnliche Performance wie Runtimes für unspezifische Programmiermodelle haben. Ziel meiner Arbeit ist es daher, Ansätze zu finden, die die Performance von vielen VMS-basierten Runtimes gleichzeitig verbessern, insbesondere mit Augenmerk auf Hardwarelösungen.

1 Ausgeführte Arbeiten

Aufgrund des Ausscheidens von Dr. Halle aus dem Fachbereich musste ich dieses Jahr die gesamte Entwicklungsarbeit für das VMS-System übernehmen. (Vorher war diese unter vier Mitgliedern des Fachgebiets geteilt.) Dies bedeutete, dass ich mich zunächst

nicht wie geplant der Portierung von VMS auf die Beefarm-Plattform[8] widmen konnte. Sowohl für meine als auch für andere Arbeiten des Fachgebiets war eine VMS-basierte Implementation des OmpSs Programmiermodells[4] eingeplant, die zwar zur Verfügung stand, aber noch erhebliche Fehler enthielt. Des weiteren waren auch einige Features noch nicht implementiert, die für die komplizierteren Benchmarks wie z.B. den H.264-Videodecoder[1] benötigt wurden.

Die Entwicklungsarbeit bis die Benchmarks problemlos ausgeführt wurden, dauerte bis März 2013. In dieser Zeit habe ich neben der Fehlerbehebung folgende Funktionalitäten zum VMS-System bzw. dem OmpSs-Plugin hinzugefügt:

- Unterstützung für den Präprozessorbefehl *#pragma omp taskwait on(*p)*, der es einem Thread oder Task erlaubt, auf die Beendung aller im System befindlichen Tasks zu warten, die die Datenstruktur p verändern.
- Unterstützung für den Präprozessorbefehl *#pragma omp critical(name)*, der Wechselseitigen Ausschluss garantiert.
- Die Möglichkeit, die Erstellung von neuen Tasks zu unterbinden, wenn bereits eine große Anzahl lauffähiger Tasks im System existiert (*throttling*).
- Automatische Erkennung von Deadlocks im Programm unabhängig vom eingesetzten Programmiermodell.
- Initialisierung und Beendung des VMS-Runtimes ohne dass diese Funktionen aus dem Programm explizit aufgerufen werden müssen.
- Die Möglichkeit, den API-Funktionen des Programmiermodells eine beliebige Signatur zu geben (Entfernung von bisherigen Pflichtargumenten). Gemeinsam mit dem vorigen Feature erlaubt dies, API-kompatible Ersatzbibliotheken für bereits existierende Programmiermodelle zu erstellen. Somit kann man existierende Benchmarks bzw. Programme für diese Modelle ohne Änderungen übernehmen.
- Ein Interface für externe Dependency Resolution-Mechanismen, wie z.B. das im Fachgebiet entwickelte Nexus++ board[3].

Des weiteren habe ich den geplanten Artikel über das letztes Jahr entwickelte Instrumentationsinterface von VMS verfasst. Der notwendige Aufwand, bis die interessanteren Benchmarks ausgeführt werden konnten, hatte jedoch zur Folge, dass ich nur einfache Beispiele vorbereiten konnte. Diese konnten nicht auf dem hohen Niveau der anvisierten Konferenz überzeugen und der Artikel wurde nicht zur Veröffentlichung angenommen. Zurzeit überarbeite ich ihn, um ihn mit zusätzlichen Beispielen erneut einzureichen.

2 Zukünftige Aktivitäten

Eine weitere Herausforderung ist der Teil des VMS-systems, der für die Kohärenz der internen Zustandsdaten des Runtimes sorgt. Die jetztige Implementierung ist ein Proof-of-Concept, in dem die Kohärenz sichergestellt wird, indem alle Runtimefunktionen über ein globales Mutex sequentialisiert werden. Diese Lösung ist sehr einfach zu implementieren und bietet unter gewissen Bedingungen nur geringfügig schlechtere Performance als eine Lösung die die einzelnen Bestandteile getrennt synchronisiert. Dies ist der Fall, solange die Anzahl Prozessorkerne im System ein Dutzend nicht überschreitet, und die Runtimeaufrufe im Vergleich zu den Berechnungen der Anwendung nur wenig Rechenzeit benötigen. Insbesondere jedoch für Anwendungen die keine größeren unabhängigen Abschnitte sondern nur sehr fein granularen Parallelismus aufweisen, ist diese Lösung suboptimal. Verschärfend kommt hinzu, das gerade in diesen Fällen parallele Programmiermodelle wünschenswert sind, die die Komplexität aus der Programmierung nehmen und sie anstattdessen in das Runtime verschieben.

Für die Entwicklung der unterstützenden Hardware für das VMS-System sind diese Einschränkungen hinnehmbar, da es keinen Einfluss auf die Funktionalität hat. Bei der Performanceevaluation hingegen würde dies zu einer Überbewertung des Effekts der Hardware führen, da diese Lösung besonders empfindlich auf die Länge der Runtimeaufrufe reagiert. Daher war ursprünglich geplant, dass ich die Hardware zunächst auf Basis der jetztigen Lösung entwickle, und dann mit der inzwischen vorangetriebenen verteilten Version von VMS integriere. Aufgrund des Ausscheidens von Dr. Halle kann nun nicht mehr erwartet werden, dass diese Version ohne mein Zutun fertiggestellt wird.

Eine verteilte Version von VMS wäre aber nicht nur für die Evaluation von spezialisierter Hardware von Nutzen, sondern würde auch die Benutzung von VMS für größere Systeme, verteilte Systeme, und Anwendungen mit niedrigem Parallelismus ermöglichen. Das Matsuoka Lab des Tokyo Institute of Technology in Japan hat Interesse bekundet, an einer verteilten Version mitzuwirken, um damit ein modulares OmpSs-runtime zu implementieren. Das Konzept dieses modularen Runtimes ist, dass nur die von der Anwendung wirklich benötigten Funktionalitäten auch geladen werden. So kann z.B. auf Tests von Konditionen verzichtet werden, die nur zutreffen können, wenn man in der Anwendung nicht vorkommende Funktionen aufruft. Dadurch wird die Rechenzeit verringert, die im Runtime verbracht wird.

Wird diese Kooperation erfolgreich abgeschlossen, eröffnen sich weitere Forschungsmöglichkeiten. Eine davon ist die automatische Anpassung von der Taskgröße an die Fähigkeiten der Hardware. Dies ist in zwei Richtungen möglich: Zum einen kann man eine Gesamtaufgabe angeben, sowie eine Methode um diese aufzuteilen und später wieder zusammenzuführen. Dieses Prinzip findet bereits Verwendung in VMS, in Form des Divider-Kernel-Undivider Patterns[6]. Zum anderen kann man die kleinstmöglichsten Tasks spezifizieren und die Zusammenführung zu größeren Einheiten dem Runtime überlassen. Dabei besteht die Komplexität darin, das die Zusammenführung kaum Overhead hinzufügen darf, da sie bei feingranularen Tasks extrem häufig stattfinden muss. Eine erfolgreiche Anwendung dieses Prinzips gibt es daher hauptsächlich für das Cilk[2] Programmiermodell, das für

den extrem niedrigen Overhead seines Runtimes bekannt ist. VMS hat ebenfalls niedrige Overheads beim Taskwechsel, wenn geeignete Plugins verwendet werden, sodass dieser Ansatz auch hier Anwendung finden könnte.

Als Teil dieser Zusammenarbeit plane ich die Monate Oktober bis März am Tokyo Institute of Technology zu verbringen. Das Matsuoka Lab ist Teil des GSIC Exzellenzzentrums für Supercomputing und ist in der Programmierung großer und verteilter Systeme spezialisiert. Ein Fokus der Forschungsgruppe ist auf neuartige Programmiermodelle. Es ergeben sich so sowohl gemeinsame Interessen als auch sich ergänzende Expertise.

Bis Ende September plane ich, eine erste alpha-Version des verteilten VMS-Systems zu implementieren, und diese dann in der ersten Zeit am Tokyo Institute of Technology zu verbessern. Darauf aufbauend werde ich ein modulares OmpSs-Plugin implementieren und anhand der Benchmarks und Wissenschaftlichen Anwendungen des Fachgebiets AES und des Matsuoka Lab evaluieren. Bei der Implementierung werde ich stets auf die Trennung der verschiedenen Bestandteile des Runtimes achten und die Abstraktionen so gestalten, dass sie sich auch als Schnittstellen zu Modulen mit Hardwareunterstützung eignen. Dadurch muss für die Integration von verschiedenen Hardwaremodulen nur sehr wenig Code modifiziert werden, und Module können einfach kombiniert und ausgetauscht werden.

Nach meiner Rückkehr werde ich einige Hardwarebeschleuniger in die neue verteilte VMS-Version einbauen. Zusätzlich zu der geplanten Unterstützung für Hashtabellen bietet diese Version neue Ansatzpunkte, insbesondere die Unterstützung der Kommunikation von Runtime-Updates zwischen Prozessorkernen.

Durch die Refokusierung auf die verteilte VMS-Version werde ich diese Hardwarelösungen nicht so umfangreich betrachten können wie ursprünglich geplant. Jedoch können VMS-basierte Runtimes wie das OmpSs-Runtime bereits jetzt vergleichbare Laufzeit mit den offiziellen Runtimes aufweisen, und 80% der zurzeit im VMS-Runtime verbrachten Zeit ist Blockierung durch das Mutex-Problem, das ich durch die verteilte VMS-Version beheben werde. Das ursprüngliche Ziel, VMS-basierte Runtimes mit handoptimierten Runtimes konkurrenzfähig zu machen, sollte also durch diese Softwarelösung bereits erreicht werden.

Zusammenfassend ergibt sich folgender Zeitplan:

Verteilte VMS-Version	Juni - September 2013	4 Monate	
Abstraktionen verfeinern	Oktober - November 2013	2 Monate	} am GSIC
Modulares OmpSs Plugin	Dezember 2013 - Januar 2014	3 Monate	} am GSIC
Evaluation	März 2014	1 Monat	} am GSIC
Hardwareintegration	April - Juni 2013	3 Monate	
Gesamtevaluation	Juli - August 2014	2 Monate	
Niederschrift	September 2013 - Februar 2014	6 Monate	

Tabelle 1: Zeitplan

Literatur

[1] M. Andersch, C. C. Chi, and B. Juurlink; *Using OpenMP Superscalar for Parallelization of Embedded and Consumer Applications*; Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation; 2012

[2] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou; *Cilk: an efficient multithreaded runtime system*; SIGPLAN Not. 30, 8; 1995

[3] T. Dallou, B. Juurlink; *Hardware-Based Task Dependency Resolution for the StarSs Programming Model*; Parallel Processing Workshops (ICPPW), 2012

[4] A. Duran, E. Ayguade, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, J. Planas; *Ompss: a proposal for programming heterogeneous multi-core architectures*; Parallel Processing Letters, 21(02); 2011

[5] S. Halle, A. Cohen; *Support of Collective Effort Towards Performance Portability*; Proceedings of 3rd USENIX Workshop on Hot Topics in Parallelism, May 2011

[6] S. Halle, A. Cohen; *The DKU Pattern for Performance Portable Parallel Programming*; 2009

[7] M. Mernik, J. Heering, A. M. Sloane; *When and how to develop domain-specific languages*; ACM Comput. Surv. 37, 4; 2005

[8] N. Sonmez, O. Arcas, G. Sayilar, O. S. Unsal, A. Cristal, I. Hur, S. Singh, M. Valero; *From plasma to beefarm: design experience of an FPGA-based multicore prototype*; Proceedings of the 7th international conference on Reconfigurable computing: architectures, tools and applications (ARC'11), 2011