Zusammenfassung

Parallelprogrammierung für Systeme mit gemeinsamem Speicher (Shared Memory) scheint auf den ersten Blick oftmals recht einfach, wie zum Beispiel das Einfügen von OpenMP Pragmas in den Programmcode. Dabei hängt die für Anwendungen erreichbare Leistung von bestimmten Eigenschaften der Systemarchitektur ab – beispielsweise die erreichbare Speicherbandbreite innerhalb der Speicherhierarchie – und in wie weit diese in der Anwendungsentwicklung berücksictigt wurden. Diese Arbeit präsentiert Lösungen um Shared Memory-parallele Anwendungen mittels eines methodischen Ansatzes für aktuelle und kommende Architekturen auszulegen. Für dieses Ziel wird eine erfolgreiche Strategie aus dem Software-Engineering eingesetzt: die Einführung von Abstraktionen.

Mit dem uneinheitlichen Speicherzugriffsverhalten auf großen Shared Memory Systemen sowie bei der Berücksichtigung von Energieeffizienz wurde der Ausdruck und die Verwaltung von Datenlokalität wichtig auf aktuellen Systemarchitekturen, mit weiter steigender Bedeutung. Weder wurden die de facto Parallelisierungsstandards MPI und OpenMP dafür ausgelegt, noch ist diese Aufgabe interessant und geeignet für Anwendungswissenschaftler. Geeignete Abstraktionen für Parallelität und Datenlokalität müssen um erfolgreich zu sein gleichzeitig leistungsfähig und einfach in der Anwendung in bestehende Programmcodes sein. Die Bedeutung des Wortes Abstraktion in dieser Arbeit ist zweifältig: zum einen ist damit die methodische Auswahl der Systemarchitektureigenschaften gemeint die wichtig sind um Anwendungsperformance zu erreichen, und zum zweiten, bezeichnet es den Entwurf von Konzepten und Softwarekomponenten für die Parallelprogrammierung und Parallelisierung von Simulationsanwendungen.

Damit die Abstraktion vom Endnutzer akzeptiert werden können, müssen vorhandene Daten- und Programmstrukturen soweit wie möglich unverändert bestehen bleiben können, insbesondere in Objekt-orientierten Programmen. Entsprechend müssen die Abstraktionen in einer allgemein verwendbaren Form ausgedrückt werden, beispielsweise müssen sie sich mit üblichen Designmustern integrieren lassen. Um dieses Ziel zu erreichen, werden in dieser Arbeit im ersten Schritt Speicherverwaltungsstrategien für NUMA Systeme identifiziert. Anschließend wird ein leistungsfähiges und dennoch einfach anzuwendendes Thread-Affinity Modell für OpenMP entwickelt. Schließlich wird für die Speicherverwaltungsstrategien und das Thread-Affinity Modell gezeigt, wie sie auch für die Parallelisierung mit Objekt-orientierten Abstraktionen geeignet sind. Um dies zu unterstützen werden

mehrere Benchmarks und Experimente gezeigt um das Verhalten von OpenMP Implementierungen zu analysieren. In ihrer Gesamtheit liefert diese Arbeit somit einen methodischen Ansatz zur Entwicklung paralleler technisch-wissenschaftlicher Software für Multi- und Many-Core-Architekturen.

Abstract

Shared memory parallel programming, for instance with OpenMP, might look simple at first sight. However, the achievable performance of real-world applications depends on certain machine properties – for instance the achievable memory bandwidth within the memory hierarchy – and in how far the application programmer has taken these into account. This thesis presents solutions for designing shared memory parallel applications targeting current and future system architectures by following a methodical approach. Therefore, it builds on a successful strategy from the software engineering discipline: the introduction of abstractions.

With large shared memory machines typically providing a non-uniform memory access, and taking energy efficiency into account, expressing and managing data locality is important today and will be even more so on the next generation of machines. The de-facto standard parallelization paradigms MPI and OpenMP were not well-equipped to allow for that, nor is this a task a domain scientist is interested in solving. Suitable abstractions for handling parallelism and data locality have to be powerful enough to fulfill their purpose while being simple enough to be applicable to existing application codes. The means of abstraction in this work are twofold: first, it relates to the methodical selection of those architecture-specific properties that are important to achieve performance, and second, it relates to the design of parallel programming model concepts and of software components to foster the parallelization of simulation codes.

For the abstractions to be acceptable by end users, existing data structures and code designs should be left unmodified as much as possible, in particular in object-oriented codes. Hence, the abstractions have to be formulated in a broadly accepted form, for instance they have to integrate well with common design patterns. To achieve this goal, this work first identifies several memory management idioms for NUMA machines. Second, a powerful yet simple-to-use and flexible thread affinity model for OpenMP is developed. Third, the memory management idioms and the thread affinity model are shown to support the parallelization with object-oriented abstractions. To support all this, a set of benchmarks and experiments to analyze OpenMP implementation behavior is presented. This work as a whole proposes a methodic approach to develop parallel scientific software for multi- and many-core architectures.