Basisfall Vergleichsbasiertes Sortieren

Programmieraufgabe Algorithm Engineering

Dennis Felsing

2013-02-07

1 Einleitung

In dieser Programmieraufgabe sollten Basisfälle für vergleichsbasiertes Sortieren untersucht werden. Dies wurde anhand von Quicksort und Super Scalar Sample Sort durchgeführt.

Alle Messungen wurden mit g++ 4.7.2 mit Optimierungsstufe `-O3` und Profile Guided Optimization durchgeführt. Die Testmaschine verfügt über eine Intel-Core2-Quad-Q9300-CPU mit 2.5 GHz. Es wurden zufällig generierte 64-bit Integer sortiert.

2 Quicksort

Für sehr kleine Arrays ($n <= 8$) bieten sich Sortiernetzwerke in Form ausprogrammierter If-Strukturen an. Da solche Fälle jedoch in den betrachteten Algorithmen selten vorkamen wurden Sortiernetzwerke nicht weiter betrachtet.

Beim Sortieren kleiner Arrays ($n < 100$) ist Insertionsort als schnellster Sortieralgorithmus bekannt. [1] Aus diesem Grund wurde Insertionsort als Grundlage für weitere Experimente verwendet.

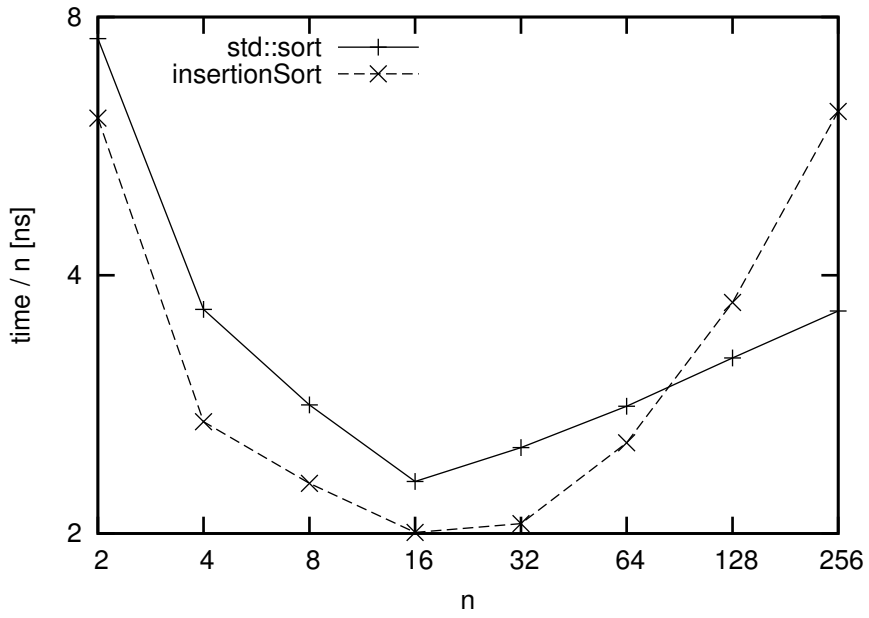

Abbildung 1: Sortieren kleiner Arrays

Eine abgewandelte Form von Insertionsort, die k Elemente auf einmal verschiebt, wurde für $k = 2$ und $k = 4$ implementiert, brachte jedoch im Bereich $n < 100$ keine besseren Ergebnisse als einfaches Insertionsort.

Somit bietet sich Insertionsort als Basisfall für Quicksort an. Experimentell wurde 64 als geeignete Grenze für den Basisfall bestimmt.

Bei Quicksort ist es möglich die Basisfälle nicht direkt zu behandeln, sondern diese erst am Ende zu sortieren. Dadurch spart man sich das ständige Springen in den Basisfall während der Quicksort-Abarbeitung. Insertionsort eignet sich besonders gut um auf diese Weise nach Abarbeiten von Quicksort das gesamte Array erneut zu sortieren. In einem teilsortierten Array verschiebt Insertionsort die Elemente nur innerhalb der einzelnen Bereiche, aber nicht über diese hinaus.

Ein Kompromiss zwischen den normalen Basisfällen und dem anschließenden Sortieren bietet ein Buffer. In diesen Buffer werden Basisfälle hinzugefügt sobald sie anfallen. Ist der Buffer voll, so werden alle vorhandenen Basisfälle abgearbeitet und der Buffer geleert. Dabei werden direkt aneinander liegende Basisfälle zusammengefasst. Wie Abbildung 2 zeigt kann jedoch für dieses Verfahren als Basisfall bei Quicksort kein Performance-Vorteil gegenüber Insertionsort am Ende festgestellt werden.

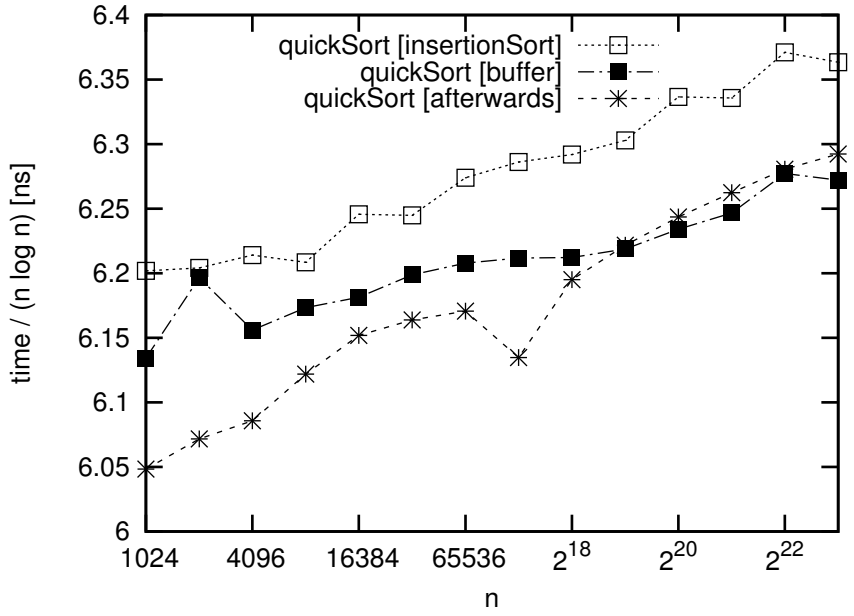

Abbildung 2: Quicksort mit verschiedenen Varianten des Basisfalls

Für Eingaben $n <= 2^{17}$ liefert Quicksort mit späterer Sortierung ein besseres Ergebnis als std::sort, wie Abbildung 3 zu entnehmen ist.

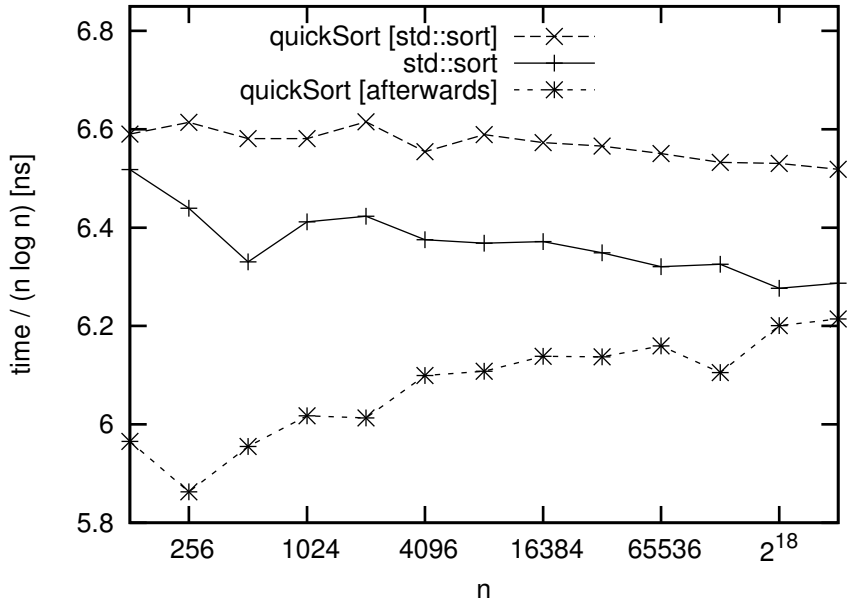

Abbildung 3: Quicksort und std::sort für mittelgroße Arrays

3 Super Scalar Sample Sort

Um für größere Eingaben effizientes Sortieren zu ermöglichen wird im folgenden Super Scalar Sample Sort betrachtet.

In Super Scalar Sample Sort lässt sich Insertionsort ebenfalls als Basisfall verwenden. Da aber weder SSSS noch Insertionsort für den Bereich $100 < n < 1024$ besonders effizient sind, ist das Endergebnis ebenfalls langsam, was sich in Abbildung 4 sehen lässt.

Wie wir bereits gesehen haben ist in diesem Bereich Quicksort performant, weshalb wir Quicksort als Basisfall für SSSS für $n < 1024$ verwenden. Quicksort selber überspringt die Basisfälle der Größe $n < 64$. Anschließend wird Insertionsort über das gesamte Array ausgeführt um die Basisfälle abzudecken. Damit wird eine bessere Performance als durch Verwenden von std::sort als Basisfall erreicht.

Aus Abbildung 4 lässt sich ablesen, dass der Basisfall bei größeren Eingaben weniger Zeit in Anspruch nimmt. Bei $n = 16384$ benötigt der Basisfall noch 68% der Laufzeit, bei $n = 2^{25}$ nur noch 35%.

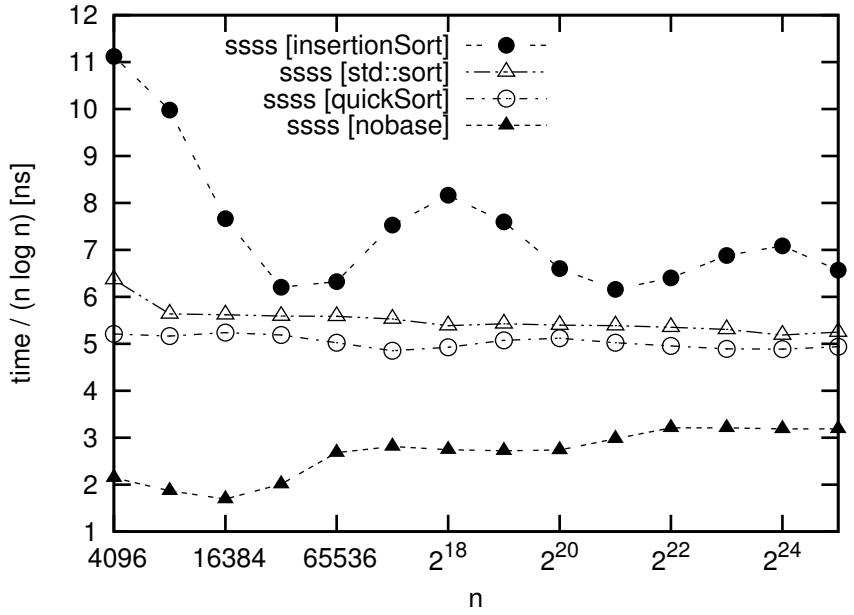

Abbildung 4: Vergleich Basisfälle für Super Scalar Sample Sort

In Abbildung 5 sind die Ergebnisse für Quicksort und SSSS im Vergleich zu std::sort zusammenfassend dargestellt.

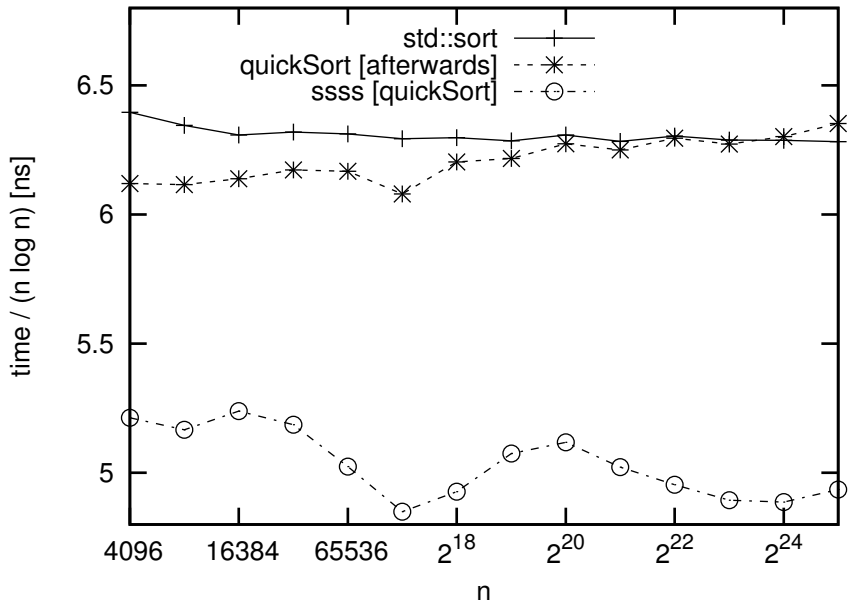

Abbildung 5: Vergleich Super Scalar Sample Sort mit anderen Sortieralgorithmen

Literatur

[1] Renato F. Werneck and Celso C. Ribeiro. Sorting methods for small arrays, 2000.