Einführung in die Parallelverarbeitung
Prof. Claudia Fohry
M.Sc. Rüdiger Nather

Universität Kassel
Wintersemester 2025/26
Abgabe 10.2. um 10.00 Uhr

Projektaufgabe MPI

Die Konditionen für die Bearbeitung der folgenden Aufgabe sind die gleichen wie bei den vorherigen Aufgabenblättern.

`bubble.c(pp)`: Wir betrachten die folgende einfache Variante von Bubblesort. Bei dieser wird das Array immer in der gleichen Richtung durchlaufen, so dass sich die großen Elemente schnell nach rechts, aber die kleinen nur langsam nach links bewegen. Zudem führt der Algorithmus stets $n-1$ Iterationen der äußeren Schleife aus, auch wenn bereits in früheren Iterationen nicht mehr getauscht wurde. In `n_swaps` wird die Anzahl der Tauschoperationen gezählt.

```
unsigned int n_swaps = 0;
for (int i = n - 1; i > 0; --i) {
  for (int j = 0; j < i; ++j) {
    if (a[j].val > a[j+1].val) {
      swap(j, j+1);
      n_swaps++;
    }
  }
}
```

Ziel der Aufgabe ist die Parallelisierung *genau dieses* Algorithmus mittels MPI.

Initialisieren Sie zunächst ein Eingabearray der Länge n. Die Initialisierung soll parallel durch $p|n$ Prozesse erfolgen, so dass jeder Prozess im Anschluss n/p Arrayelemente abspeichert: P_0 die linkesten, P_1 die nächsten etc. Insgesamt muss das Array die gleichen Elemente wie nach der folgenden sequentiellen Initialisierung enthalten. Wie im Code zu erkennen, ist jedes Arrayelement eine `struct`, die einen zufälligen `double`-Wert und einen Index vom Typ `int` enthält:

```
for (int i = 0; i < n; ++i) {
  srand(seed * (i+5));
  a[i].nr = i;
  a[i].val = (rand() % 100) / 10.0;
}
```

WICHTIG: Die parallele Variante des Algorithmus muss exakt die gleichen Vergleiche und Tauschoperationen ausführen wie die sequentielle. Folglich muss sich der gleiche Wert für `n_swaps` ergeben. Welcher Prozess welche der Operationen ausführt und wann, ist dagegen Ihnen überlassen. Es ist auch zulässig, Daten zu replizieren und die gleichen Operationen mehrfach auszuführen. In diesem Fall werden sie bei `n_swaps` nur einmal gezählt. Zusätzlich zu $p|n$ dürfen Sie weitere Annahmen zur Größe von n und p treffen. Geben Sie diese ggf. bei Programmaufruf in einer Ausschrift an und brechen Sie das Programm bei unzulässigen Eingaben ab.

Nach dem Sortieren muss das Array wieder gleichmäßig auf die p Prozesse verteilt sein, so dass P_0 die linkesten n/p Elemente enthält, P_1 die nächsten etc. Wie das Array zwischenzeitlich verteilt ist, ist Ihnen überlassen.

Alle Entwurfsentscheidungen sollen so getroffen werden, dass das Programm, vor allem auf großen n, *möglichst effizient* läuft. Achten Sie insbesondere auf die Realisierung der Kommunikation.Von der Forderung nach Effizienz ausgenommen ist eine nicht-optimale Lastenbalancierung, die sich (vermutlich) daraus ergibt, dass große Elemente schnell nach rechts aufsteigen und somit Prozesse mit großen Nummern früher ihre Arbeit beenden als solche mit kleinen. Hierzu müssen Sie *nichts* implementieren. Überlegen Sie sich jedoch einen Ansatz für den Umgang mit diesem Problem und skizzieren Sie diesen kurz in einem Kommentar. Sollten Sie weitere Optimierungsideen haben, deren Implementierung zu aufwändig wäre, können Sie auch diese in einem Kommentar skizzieren. In beiden Fällen geht es *nur* um Optimierungen, die den (ineffizienten) sequentiellen Algorithmus unverändert lassen.

Das Programm muss sich mittels

```
mpirun -n <n> ./bubble n seed
```

aufrufen lassen. Dahinter können Sie bei Bedarf weitere Parameter benutzen. Erklären Sie diese ggf. bei Programmaufruf in einer Ausschrift. Das Programm soll folgendes ausgeben:

- `n_swaps`: Anzahl der Tauschoperationen (siehe oben). Im Falle eines Überlaufs kann der Wert negativ sein.
- Für jeden der Prozesse das linkeste und rechteste der nach dem Sortieren an diesem Prozess gespeicherten Arrayelemente. Die Ausgabe muss nach Prozessnummern geordnet sein. Die Ausgabe darf verteilt oder vollständig durch P_0 erfolgen. (Bei der Ausgabe ist die Effizienz egal.)
- Für $n \leq 20$: Inhalte von Ein- und Ausgabearray in sortierter Reihenfolge. Auch diese Ausgabe darf verteilt oder vollständig durch P_0 erfolgen.

Beispiel:

```
mpirun -n 4 ./bubble 20 123

Eingabe: (0, 4.8)   (1, 3.5)   (2, 6.9)   (3, 5.6)   (4, 0.0)   (5, 5.4)
         (6, 5.3)   (7, 2.4)   (8, 0.1)   (9, 9.4)   (10, 0.3)   (11, 1.6)
         (12, 6.6)   (13, 5.8)   (14, 7.8)   (15, 5.2)  (16, 2.7)
         (17, 8.4)   (18, 8.4)   (19, 4.4)

n_swaps = 79

P0: (4, 0.0)  (7, 2.4)
P1: (16, 2.7)  (15, 5.2)
P2: (6, 5.3)  (12, 6.6)
P3: (2, 6.9)  (9, 9.4)

Ausgabe: (4, 0.0)   (8, 0.1)   (10, 0.3)   (11, 1.6)   (7, 2.4)   (16, 2.7)
         (1, 3.5)   (19, 4.4) (0, 4.8)   (15, 5.2)   (6, 5.3)   (5, 5.4)
         (3, 5.6)   (13, 5.8)   (12, 6.6)   (2, 6.9)  (14, 7.8)   (17, 8.4)
         (18, 8.4)   (9, 9.4)
```

Abgabe: Alle Lösungen müssen bis spätestens 10.2.2026, 10:00 Uhr per Email an fohry@uni-kassel.de mit dem Betreff “IntroPV Abgabe“ abgegeben werden. Später abgegebene Lösungen werden als **nicht abgegeben** gewertet!