9 Sortieren

- $m = nP$ Eingabewerte. Anfangs n pro PE
- u.U. allgemeiner
- Ausgabe global sortiert

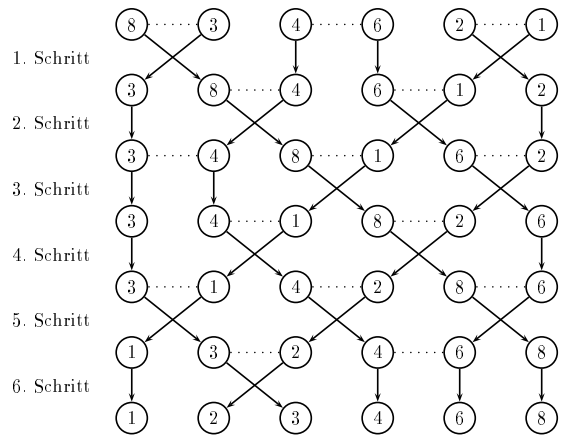
$$\boxed{d_{0,0},\ldots,d_{0,n-1}} \quad , \ldots , \quad \boxed{d_{P-1,0},\ldots,d_{P-1,n-1}}$$

$$\Downarrow \pi$$

$$\boxed{s_{0,0} \le \cdots \le s_{0,n_1-1}} \le \cdots \le \boxed{s_{P-1,0} \le \cdots \le s_{P-1,n_{P-1}-1}}$$

- Vergleichsbasiertes Modell
- $T_{\text{seq}} = T_{\text{compr}} m \log m + \mathrm{O}(m)$

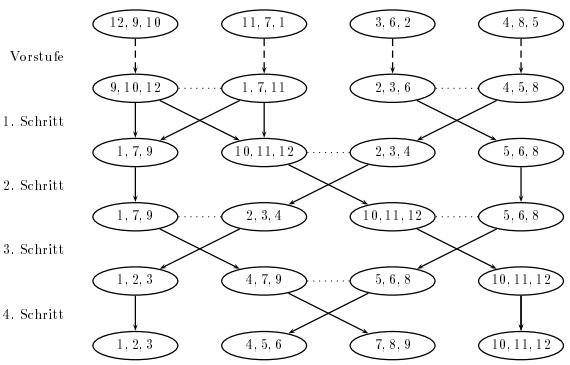
9.1 Odd-Even-Sort ($n = 1$)



9.1.1 Odd-Even-Sort (beliebige n)

+ Spezieller Zellularautomat
– $\Theta(mn)$ Vergleiche pro PE
– $\Theta(P)$ bis $\Theta(m)$ Startups
⇒ Nur als Hardwarealgorithmus für kleine n und P (vielleicht) von Interesse

9.1.2 Merge-Splitting-Sort



+ Lokal läßt sich effizienter Sortieralgorithmus einsetzen: $\Theta(n \log n)$ Vergleiche
– P Startups
– $\Theta(m)$ Kommunikation pro PE
– Immer noch $\Theta(m)$ Vergleiche für merging
– Effizient nur für $n \gg 2^P T_{\text{byte}}/T_{\text{compr}}$

9.2 Sortiernetzwerke

- Netzwerk Schaltelemente, die Compare-Exchange machen

\+ Einfach

- Bitonic-Sort, ... : Tiefe $O(\log^2 m)$ Größe $O(m \log^2 m)$
- AKS-Sortiernetzwerk: Tiefe $c \log m$, Größe $O(m \log m)$ $c \gg 1000$, ...

9.3 Schnelles Ranking ($P = m^2$)

- $m \times m$ logisches Gitter
- Eingabe in PEs (i, i)
- a = Broadcast Spalte i
- b = Broadcast Zeile i
- $x = [a < b]$
- Rang von Element i ist $\sum_j x_{ij}$

\+ $O(\log P)$ Startups

– $\Theta(m^2)$ Vergleiche insgesamt

9.4 Sample-Sort

Splitter fallen vom Himmel

{Für $0 < k < P$ sei p_k das Element mit Rang $k \cdot n$.}
{Außerdem setzen wir $p_0 = -\infty$ und $p_P = \infty$.}
initialisiere P leere Nachrichten N_k,
$(0 \leq k < P)$;
for $i \leftarrow 0$ $n-1$ **do**
  bestimme k, so daß $p_k \leq d_i < p_{k+1}$;
  nimm d_i in Nachricht N_k auf;
end for
for $k \leftarrow 0$ $P-1$ **do**
  schicke Nachricht N_k an PE k;
end for
empfange P Nachrichten;
sortiere empfangene Daten;

9.4.1 Eine Variante von All-to-all

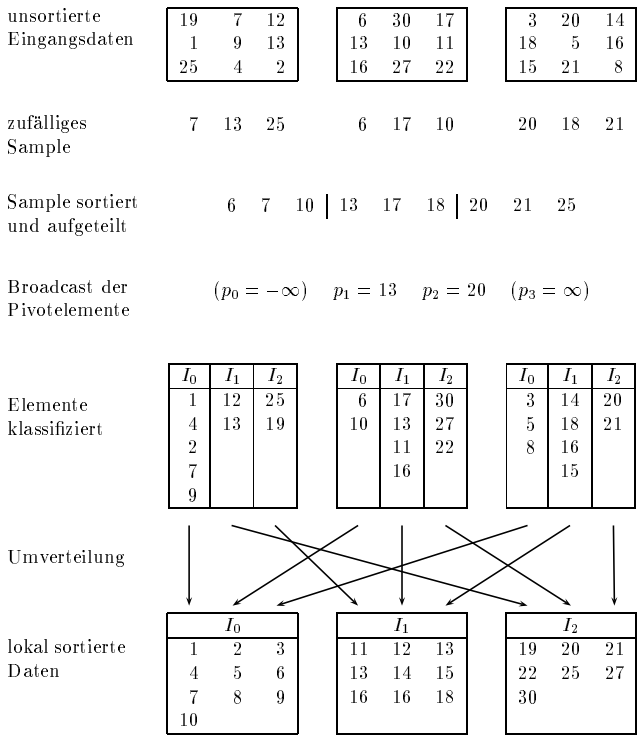
```
int my_Alltoallv(void *sendbuf, int *sendcounts,
                 int *sdispls, MPI_Datatype sendtype,
                 void *recvbuf, int maxrcount,
                 int *size, int *recvcounts,
                 int *rdispls, MPI_Datatype recvtype,
                 MPI_Comm comm)
{  int numProcs, i;
   MPI_Comm_size(comm, &numProcs);
   MPI_Alltoall(sendcounts, 1, MPI_INT,
                recvcounts, 1, MPI_INT, comm);
   rdispls[0] = 0;
   for(i = 1;  i < numProcs;  i++) {
      rdispls[i] = rdispls[i-1] +
                   recvcounts[i-1];
   }
   *size = rdispls[numProcs-1] +
           recvcounts[numProcs-1];
   return MPI_Alltoallv(sendbuf, sendcounts,
                        sdispls, sendtype,
                        recvbuf, recvcounts,
                        rdispls, recvtype, comm);
}
```

9.4.2 Samples

choose a total of SP random elements s_k;
(S per PE) $(0 \le k < SP)$;
sort $[s_0, \ldots, s_{SP-1}]$;
for $i \leftarrow 1$ $P-1$ **do**
  $p_i \leftarrow s_{mi}$;
end for
$p_0 \leftarrow -\infty$; $p_P \leftarrow \infty$;

Satz 1. *$S \in \mathrm{O}\left(\frac{\log P}{\epsilon^2}\right)$ genügt damit kein PE mehr als $(1+\epsilon)n$ Elemente erhält.*

Ein Beispiel



9.4.3 Samples Sortieren

- Mit `MPI_(All)Gather`
- Schnelles Ranking
- Paralleles Quicksort
- Rekursiv mit Sample-Sort

9.4.4 Multi-Merging

- Vorher Sortieren
- Verbesserte Pivot-wahl
- Umverteilung
- P-Wege-Mischen

9.4.5 Looser Trees

9.5 Blick über den Tellerrand

- Datenbanken → parallele Platten
- Geometrische Datenverarbeitung
- Partikelsimulation
- Datenstrukturen: Suchbäume, Prioritätslisten