UNIVERSITÄT HAMBURG
FACHBEREICH INFORMATIK

Praktikum „Parallele Programmierung“

Travelling Salesman Problem (TSP)

von Ihar Aleinikau

1. Problemstellung

Aufgabe des TSP (auch des Rundreiseproblems):

Ein Handlungsreisender soll, ausgehend von einer Stadt, weitere Städte (n-1-Stück) genau einmal durchreisen und am Ende wieder in die Ausgangsstadt zurückkehren. Gesucht ist die Reihenfolge, in der der Handlungsreisende die n Städte besuchen muss, damit die Reise möglichst kurz ist.

2. Lösungsansatz

Exakte Lösung:

Die offensichtlich einfachste Lösung besteht in der Erzeugung aller möglichen Rundreisen, wobei am Ende die kürzeste gewählt wird. Da stets die erste Stadt konstant bleibt, beträgt die Anzahl der Permutationen bei n Städten gerade (n-1)!

Somit gibt es bei „n“ Städten $\frac{(n-1)!}{2}$ Möglichkeiten

Beispiel: Bei 15 Städte über 43 Milliarden Möglichkeiten

2. Lösungsansatz

Konzept:

- Betrachtet n Städte, angenommen $n_{max} = 25$
- Reisende anläuft jede Stadt j, abgesehen von der Ausgangsstadt, genau einmal
- Reisende verläßt jede Stadt i, abgesehen von der Ausgangsstadt, genau einmal
- Entfernungen zwischen den Städten a_{ij} werden in Form einer Matrix erfasst.
- Gesucht die kürzeste Reisestrecke.

2. Lösungsansatz

Lösung:

- Eine Karte von genmap zu generieren
- Definieren von Coordinator und Worker: Zyklen mit der Erwartung von Nachrichten

```
void Coordinator()
{
    MPI_Status status;
    Msg_t msg;

    int* waiting = new int[NumProcs];
    int nwait = 0;
    int bpath = 0;
```

```
Path Shortest;
    LIST queue;
    Path *path = new Path;
    queue.Insert(path, 0);
Shortest.length = INT_MAX;

    debug("Coordinator started");

    while (nwait < NumProcs-1) –
```

Hauptbedingung in Coordinator

2. Lösungsansatz

Lösung:

```
// Blocking receive
    MPI_Recv(&msg, MSGSIZE, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
    debug("Coordinator got message: tag=%d, size=%d", status.MPI_TAG,
MSGSIZE);
```

Ausführen den Zyklus bis alle Worker fertig sind.

2. Lösungsansatz

Lösung:

```
void Worker() {
    MPI_Status status;
    Msg_t msg;
    int shortestLength = INT_MAX;

    debug("Worker %d started", myrank);

    // Request path tag
    MPI_Send(NULL, 0, MPI_INT, 0,
GET_PATH_TAG, MPI_COMM_WORLD);

    while(1) {
    // Receive
    MPI_Recv(&msg, MSGSIZE, MPI_INT, 0,
MPI_ANY_TAG, MPI_COMM_WORLD,
&status);
```

```
 // Process done tag
    if (status.MPI_TAG == DONE_TAG) {
        debug("Worker %d received
DONE_TAG", myrank);
        break;
    }
// Update best path tag
    if (status.MPI_TAG ==
UPDATE_BEST_PATH_TAG) {
        debug("Worker %d received
UPDATE_BEST_PATH_TAG to %d", myrank,
msg.length);
        shortestLength = msg.length;
        continue;
```

2. Lösungsansatz

Lösung:

Berechnen die kürzeste Rundreise:

```
   case BEST_PATH_TAG:
       if (msg.length < Shortest.length) {
           bpath++;
           debug("Got best path %d, source = %d, length = %d", bpath,
status.MPI_SOURCE, msg.length);

           Shortest.Set(msg.length, msg.city, NumCities);
           for(int i = 1;i < NumProcs;i++)
           MPI_Send(&(Shortest.length), 1, MPI_INT, i, UPDATE_BEST_PATH_TAG,
MPI_COMM_WORLD);
       }
       break;
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
root@debian:/home/wirox/Downloads/tsp# ./genmap.sh 5 |mpirun -np 5 ./tsp -d
DEBUG: Started worker 0
DEBUG: Fill dist called from rank 0
DEBUG: Started worker 3
Number of cities: 5
  48  13  12  24  47
  23  23  16  67  38
  32  61  27  25  37
  90  29  56  50  53
  23  64  40  36  85
DEBUG: Coordinator started
DEBUG: Fill dist called from rank 3
DEBUG: Coordinator got message: tag=0, size=27
DEBUG: Worker 3 has obtained number of cities: 5
DEBUG: Coordinator got message: tag=1, size=27
DEBUG: Worker 3 has obtained dist matrix
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 3 started
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Started worker 1
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 4 started
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Worker 1 has obtained dist matrix
DEBUG: Worker 1 started
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Got best path 1, source = 4, length = 130
DEBUG: Coordinator got message: tag=4, size=27
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 1 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Worker 3 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Worker 4 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Got best path 2, source = 3, length = 127
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Worker 1 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Worker 3 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Worker 4 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Started worker 2
DEBUG: Fill dist called from rank 2
DEBUG: Coordinator got message: tag=0, size=27
DEBUG: Worker 2 has obtained number of cities: 5
DEBUG: Coordinator got message: tag=1, size=27
DEBUG: Worker 2 has obtained dist matrix
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 2 started
DEBUG: Worker 2 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Worker 2 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Coordinator got message: tag=4, size=27
Shortest path:DEBUG: Worker 2 received DONE_TAG
DEBUG: Worker 2 finished
DEBUG: Worker 3 received DONE_TAG
DEBUG: Worker 3 finished

  0  2  3  1  4; length = 127
DEBUG: Worker 4 received DONE_TAG
DEBUG: Worker 4 finished
DEBUG: Worker 1 received DONE_TAG
DEBUG: Worker 1 finished
root@debian:/home/wirox/Downloads/tsp#
```

4. Fazit

- Somit habe ich gezeigt, wie das Rundreiseproblem in parallel ausgeführten Teilprozessen gelöst wird:
- Alle beteiligten Prozesse tauschen sich gegenseitig mit Nachrichten aus
- Jeder Prozess erhält eine für ihn explizit bestimmte Nachricht

- MPI ist Spezifikation zur Datenübertragung auf nachrichtengekoppelten Systemen

- Die Kommunikation erfolgt über Kommunikatoren innerhalb von bzw. zwischen Prozessgruppen


Vielen Dank für eure
Aufmerksamkeit!

UNIVERSITÄT HAMBURG
FACHBEREICH INFORMATIK

Praktikum „Parallele Programmierung“

Travelling Salesman Problem (TSP)

von Ihar Aleinikau

1. Problemstellung

Aufgabe des TSP (auch des Rundreiseproblems):

Ein Handlungsreisender soll, ausgehend von einer Stadt, weitere Städte (n-1-Stück) genau einmal durchreisen und am Ende wieder in die Ausgangsstadt zurückkehren. Gesucht ist die Reihenfolge, in der der Handlungsreisende die n Städte besuchen muss, damit die Reise möglichst kurz ist.

2. Lösungsansatz

Exakte Lösung:

Die offensichtlich einfachste Lösung besteht in der Erzeugung aller möglichen Rundreisen, wobei am Ende die kürzeste gewählt wird. Da stets die erste Stadt konstant bleibt, beträgt die Anzahl der Permutationen bei n Städten gerade (n-1)!

Somit gibt es bei „n“ Städten $\frac{(n-1)!}{2}$ Möglichkeiten

Beispiel: Bei 15 Städte über 43 Milliarden Möglichkeiten

2. Lösungsansatz

Konzept:

- Betrachtet n Städte, angenommen $n_{max} = 25$
- Reisende anläuft jede Stadt j, abgesehen von der Ausgangsstadt, genau einmal
- Reisende verläßt jede Stadt i, abgesehen von der Ausgangsstadt, genau einmal
- Entfernungen zwischen den Städten a_{ij} werden in Form einer Matrix erfasst.
- Gesucht die kürzeste Reisestrecke.

2. Lösungsansatz

Lösung:

- Eine Karte von genmap zu generieren
- Definieren von Coordinator und Worker: Zyklen mit der Erwartung von Nachrichten

```
void Coordinator()
{
   MPI_Status status;
   Msg_t msg;

   int* waiting = new int[NumProcs];
   int nwait = 0;
   int bpath = 0;
```

```
Path Shortest;
   LIST queue;
   Path *path = new Path;
   queue.Insert(path, 0);
Shortest.length = INT_MAX;

   debug("Coordinator started");

   while (nwait < NumProcs-1) –
```

Hauptbedingung in Coordinator

2. Lösungsansatz

Lösung:

```
// Blocking receive
    MPI_Recv(&msg, MSGSIZE, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
    debug("Coordinator got message: tag=%d, size=%d", status.MPI_TAG,
MSGSIZE);
```

Ausführen den Zyklus bis alle Worker fertig sind.

2. Lösungsansatz

Lösung:

```
void Worker() {
   MPI_Status status;
   Msg_t msg;
   int shortestLength = INT_MAX;

   debug("Worker %d started", myrank);

   // Request path tag
   MPI_Send(NULL, 0, MPI_INT, 0,
GET_PATH_TAG, MPI_COMM_WORLD);

   while(1) {
   // Receive
   MPI_Recv(&msg, MSGSIZE, MPI_INT, 0,
MPI_ANY_TAG, MPI_COMM_WORLD,
&status);
```

```
 // Process done tag
   if (status.MPI_TAG == DONE_TAG) {
      debug("Worker %d received
DONE_TAG", myrank);
      break;
   }
// Update best path tag
   if (status.MPI_TAG ==
UPDATE_BEST_PATH_TAG) {
      debug("Worker %d received
UPDATE_BEST_PATH_TAG to %d", myrank,
msg.length);
      shortestLength = msg.length;
      continue;
```

2. Lösungsansatz

Lösung:

Berechnen die kürzeste Rundreise:

```
   case BEST_PATH_TAG:
       if (msg.length < Shortest.length) {
           bpath++;
           debug("Got best path %d, source = %d, length = %d", bpath,
status.MPI_SOURCE, msg.length);

           Shortest.Set(msg.length, msg.city, NumCities);
           for(int i = 1;i < NumProcs;i++)
           MPI_Send(&(Shortest.length), 1, MPI_INT, i, UPDATE_BEST_PATH_TAG,
MPI_COMM_WORLD);
       }
       break;
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
root@debian:/home/wirox/Downloads/tsp# ./genmap.sh 5 |mpirun -np 5 ./tsp -d
DEBUG: Started worker 0
DEBUG: Fill dist called from rank 0
DEBUG: Started worker 3
Number of cities: 5
  48  13  12  24  47
  23  23  16  67  38
  32  61  27  25  37
  90  29  56  50  53
  23  64  40  36  85
DEBUG: Coordinator started
DEBUG: Fill dist called from rank 3
DEBUG: Coordinator got message: tag=0, size=27
DEBUG: Worker 3 has obtained number of cities: 5
DEBUG: Coordinator got message: tag=1, size=27
DEBUG: Worker 3 has obtained dist matrix
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 3 started
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Started worker 1
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 4 started
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Worker 1 has obtained dist matrix
DEBUG: Worker 1 started
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Got best path 1, source = 4, length = 130
DEBUG: Coordinator got message: tag=4, size=27
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 1 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Worker 3 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Worker 4 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Got best path 2, source = 3, length = 127
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Worker 1 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Worker 3 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Worker 4 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Coordinator got message: tag=3, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=2, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Coordinator got message: tag=4, size=27
DEBUG: Started worker 2
DEBUG: Fill dist called from rank 2
DEBUG: Coordinator got message: tag=0, size=27
DEBUG: Worker 2 has obtained number of cities: 5
DEBUG: Coordinator got message: tag=1, size=27
DEBUG: Worker 2 has obtained dist matrix
```

3. Ergebnisse

Beispiel: n=5 mit 5 parallelen Prozessen (Coordinator und 4 Workers)

```
DEBUG: Worker 2 started
DEBUG: Worker 2 received UPDATE_BEST_PATH_TAG to 130
DEBUG: Worker 2 received UPDATE_BEST_PATH_TAG to 127
DEBUG: Coordinator got message: tag=4, size=27
Shortest path:DEBUG: Worker 2 received DONE_TAG
DEBUG: Worker 2 finished
DEBUG: Worker 3 received DONE_TAG
DEBUG: Worker 3 finished

 0  2  3  1  4; length = 127
DEBUG: Worker 4 received DONE_TAG
DEBUG: Worker 4 finished
DEBUG: Worker 1 received DONE_TAG
DEBUG: Worker 1 finished
root@debian:/home/wirox/Downloads/tsp#
```

4. Fazit

- Somit habe ich gezeigt, wie das Rundreiseproblem in parallel ausgeführten Teilprozessen gelöst wird:
- Alle beteiligten Prozesse tauschen sich gegenseitig mit Nachrichten aus
- Jeder Prozess erhält eine für ihn explizit bestimmte Nachricht

- MPI ist Spezifikation zur Datenübertragung auf nachrichtengekoppelten Systemen

- Die Kommunikation erfolgt über Kommunikatoren innerhalb von bzw. zwischen Prozessgruppen


Vielen Dank für eure
Aufmerksamkeit!