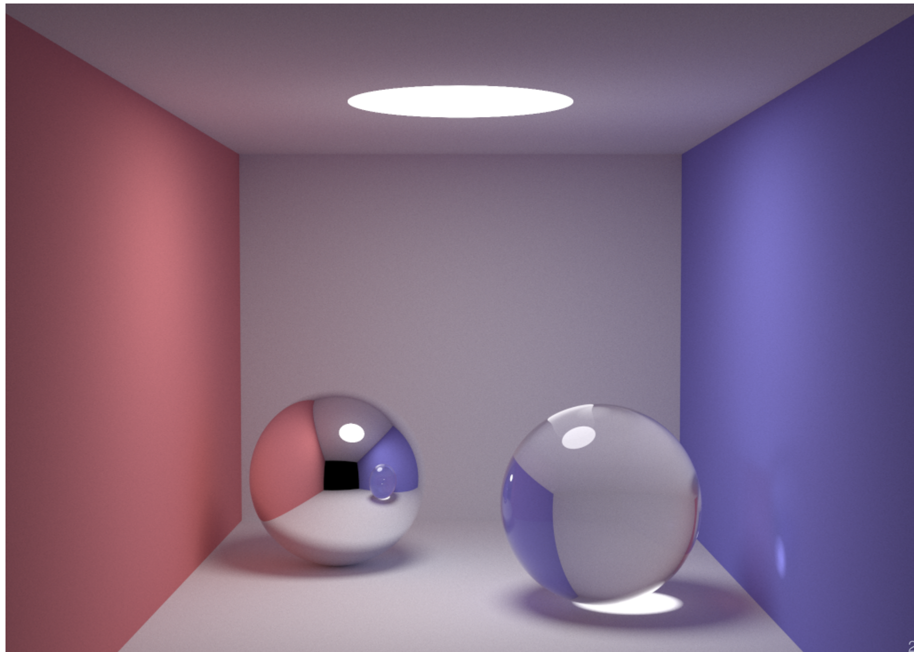
Paralleles Raytracing

Praktikum Paralleles Programmieren

Dennis Steinhoff und Alexander Koglin

24. Oktober 2014 (überarbeitete Fassung)

Szene: Cornell-Box



Raytracing

- Erzeugung eines Bildes aus einer 3D-Szene durch Strahlenaussenden von der Betrachtungsebene
- Integrieren der Lichtintensität und Farbe beim Auftreffpunkt der Strahlen
- Es lassen sich so relativ realistische Bilder erzeugen, doch dies ist sehr teuer
- hier: Szene setzt sich vollständig aus Kugeln zusammen!
- aufbauend auf smallpt von Kevin Beason

Parallelisierung

MPI:

- festes Aufteilen der Pixel
  - abwechselnd zeilenweise
  - blockweise
- Load-Balancing: dynamisches Aufteilen der Pixel/Zeilen
- optional: zufälliges Aufteilen der Pixel

Load-Balancing zur Laufzeit

Wir haben zwei potentielle Möglichkeiten betrachtet:

- Master-Slave-Warteschleife
- Knoten-zu-allen-Knoten-Kommunikation

Master-Slave-Warteschleife:

- Der Master-Prozess weist Rechenaufgaben den Knoten/Slaves zu.
- „First Come, First Serve“
- Nach dem Lösen fordert jeder Slave individuell neue Aufgaben an.

- Da die Ergebnis-Vektoren eines Slaves disjunkt zu Vektoren anderer Slaves sind, kann man per MPI_Reduce am Ende die Vektoren vom Master einsammeln.
- Dies geschieht durch Addieren der Arrays.
- Wir müssen aufgrund der Vektoren allerdings eine eigene Vektordatenstruktur und eine eigene Reduce-Funktion definieren (auch bei der festen Aufteilung).

```
void myfunc( Vec *in, Vec *inout, int *len,
                          MPI_Datatype *dptr ) {
int i; Vec c;

for (i=0; i< *len; ++i) {
        c.x = inout->x + in->x;
        c.y = inout->y + in->y;
        c.z = in->z + inout->z;
        *inout = c; in++; inout++;
}

}
```

```
MPI_Op myOp;
MPI_Datatype ctype;

/* Vektor*/
MPI_Type_contiguous( 3, MPI_DOUBLE, &ctype );
MPI_Type_commit( &ctype );
/* benutzerdefinierte Funktion */
MPI_Op_create((MPI_User_function *)&myfunc, 1, &myOp )
MPI_Barrier(MPI_COMM_WORLD);
```

```
if (rank == MASTER) {
        int row, count, sender;
        count = 0;
    for (destination = 0; destination < world_size; de
       if (destination != rank) {
          buffer[0] = count;
          MPI_Send(int_buffer, 1, MPI_INT, destination
                   MPI_COMM_WORLD);
          count = count + 1;
         }
    }
    for (i = 0; i < h; i++) {
       MPI_Recv (buffer, BUFSIZ, MPI_INT,
         MPI_ANY_SOURCE, MPI_ANY_TAG,
         MPI_COMM_WORLD, &status);
       sender = status.MPI_SOURCE;
       row = status.MPI_TAG;
       if (count < h) {
         buffer[0] = count;
```

```
    if (count < h) {
      buffer[0] = count;
      MPI_Send(buffer, 1, MPI_INT,
      sender, count, MPI_COMM_WORLD);
      count = count + 1;
      }
    else {
      MPI_Send(NULL, 0, MPI_INT, sender, h, MPI_COM
      }
   }
}
```

```
else { // Slave-Prozess
    int sum, row;
    MPI_Recv(buffer, 1, MPI_INT, MASTER,
     MPI_ANY_TAG, MPI_COMM_WORLD, &status);

    while (status.MPI_TAG !=h) { /* noch nicht ferti
      row = status.MPI_TAG;
              {
              // hier Eintrag von c berechnen....
              }

      buffer[0] = 4;
      MPI_Send (int_buffer, 1, MPI_INT, MASTER_RANK,
       row, MPI_COMM_WORLD);
      MPI_Recv (int_buffer, 1, MPI_INT, MASTER_RANK,
       MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    }
}
MPI_Reduce( c, answer, w*h, ctype, myOp, root, MPI_COM
```

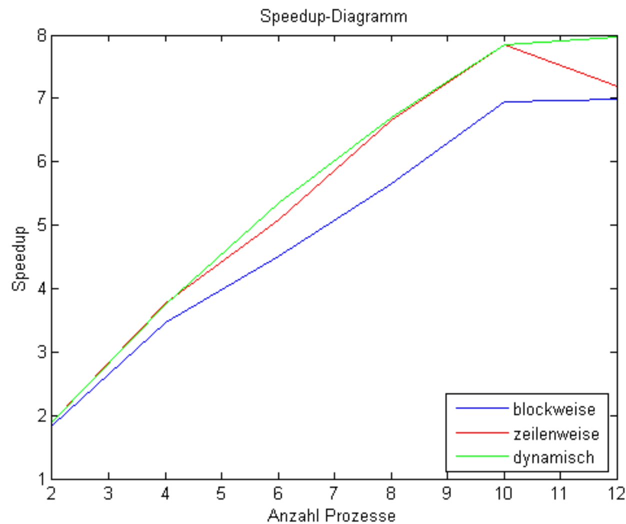
Ergebnisse

Vergleichswert bei serieller Ausführung: **102s**

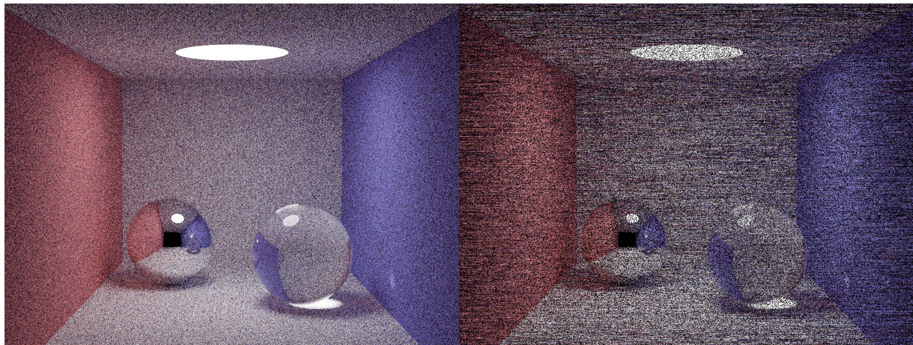
Parallelisierung	Anzahl Threads					
	2	4	6	8	10	12
Block	55,7	29,4	22,6	18,0	14,7	14,6
Zeile	53,5	27,0	20,1	15,3	13,0	14,2
Warteschleife	53,5	27,2	19,1	15,2	13,0	12,8

- Die Master-Slave-Warteschleife erzielt die besten Ergebnisse, wenn man nur die Slaves als Prozesse zählt.
- Darauf folgt die feste Zeilenaufteilung.
- Load-Balancing kann bei inhomogener Hardware-Leistung Vorteile erzielen

fast linearer Speedup



Gegenüberstellung Rechenzeiten und gerenderte Szene



Ausblick: Optimierungen

- OpenMP: for-Schleifen-Parallelisierung der Spaltenelemente
- Verwendung von SSE
- Verwendung des Intel Compilers (bei Intel CPUs)

Fazit

- Raytracing lässt sich sehr gut mit MPI parallelisieren
- Wir haben einen fast linearen Geschwindigkeitszuwachs mit zunehmender Prozessorzahl beobachtet
- Die Parallelisierung erfolgte per for-Schleifen-Parallelisierung
- Die Warteschleife ist leicht schneller als die feste abwechselnde Aufteilung.
- Bei anderen Szenen kann die Schleife noch besser sein.