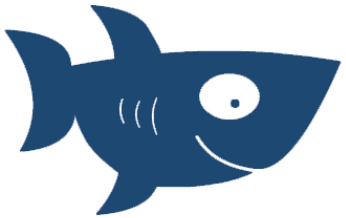
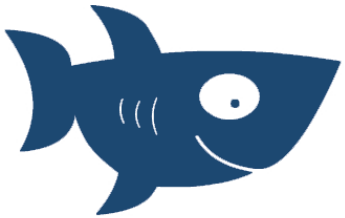


Fish & Shark

Parallele Programmierung

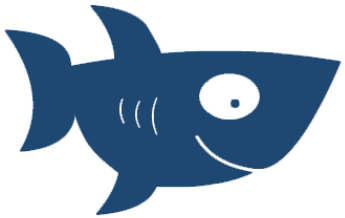
Praktikum

SS 2012



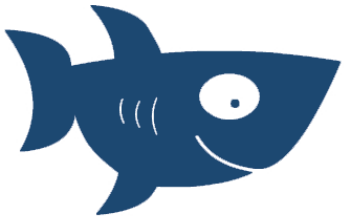
Gliederung

- Konzept
- Konzept der Parallelisierung
- Datenmodell
- Code Snippets
- Analyse
- Visualisierung
- Aufgetretene Probleme



Das Konzept

- 2 dimensionales Simulationsfeld
- Fische entstehen an zufälligen Stellen auf dem Spielfeld
- Haie fressen Fische und vermehren sich dann
- Fische und Haie bewegen sich auf dem Spielfeld (Nicht diagonal)
- Plankton entsteht zu Beginn der Simulation und vergrößert sich dann
- Haie und Fische haben eine Lebensdauer



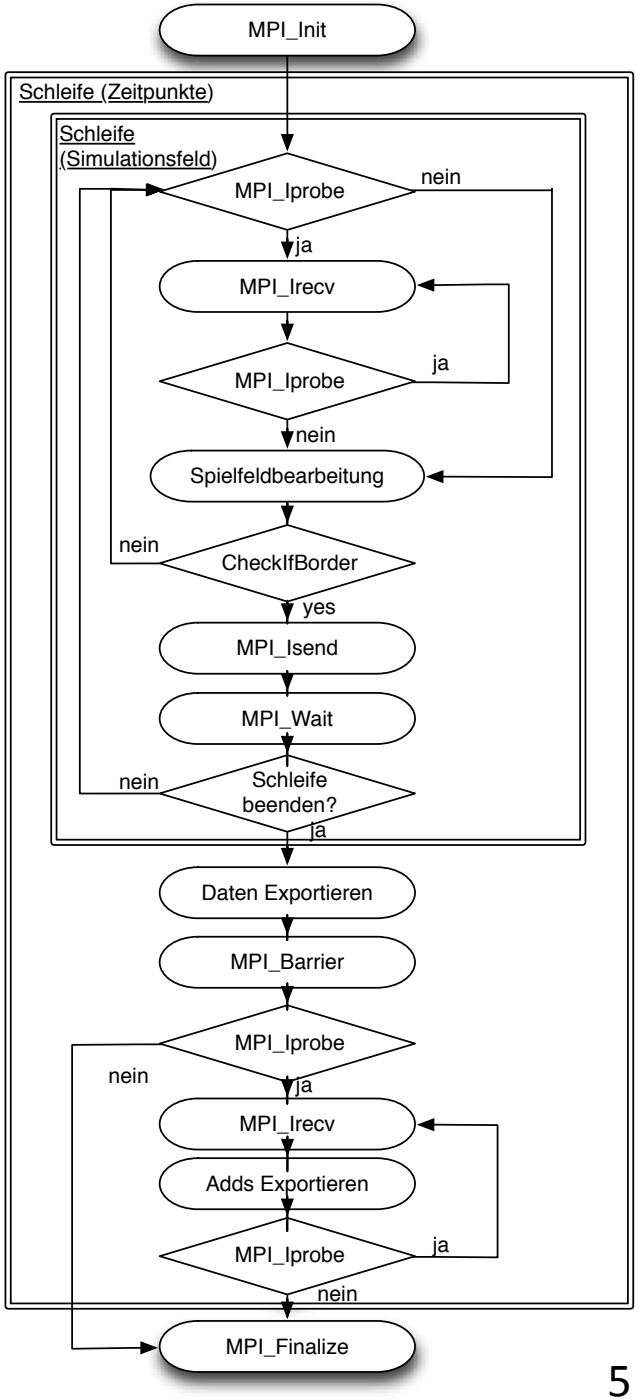
Konzept der Parallelisierung

Process 1	**Process 2**	**Process 3**	**Process 4**
Process 5	**Process 6**	**Process 7**	**Process 8**

Beispiel: 8 Prozesse

Fish & Shark

Konzept der Parallelisierung (2)

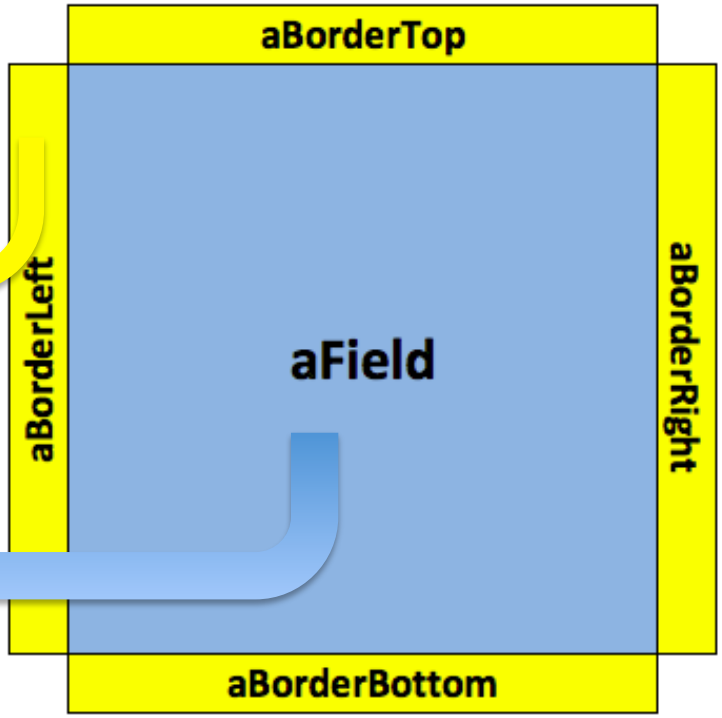


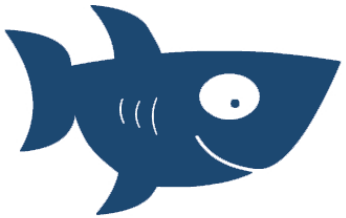
Datenmodell

aField[y][x][0-2]
0: Typ des Elements
1: Sterbe/Vermehrzeit
2: Touched this round

4 Arrays um den Zustand der anderen Prozesse zu speichern

2 dimensionales Spielfeld





Initialisierung der Arrays

```
564     // Initialize SimulationField
565     int ***aField;
566
567     aField = malloc(sizeof(int *) * iFieldSizeY);
568
569     for (int y=0;y<iFieldSizeY;y++)
570     {
571         aField[y] = malloc(sizeof(int *) * iFieldSizeX);
572
573         // Set all values to 0
574         for (int x=0;x<iFieldSizeX;x++)
575         {
576             // Initialize 3 int-Values for each Field-Position
577             // Position 0 := Value of Sector
578             // Position 1 := DieTime
579             // Position 2 := Touched this round
580             aField[y][x] = malloc(sizeof(int) * 3);
581             aField[y][x][0] = 0;
582             aField[y][x][1] = 0;
583             aField[y][x][2] = 0;
584         }
585     }
```

aField[y][x][0-2]

0: Typ des Elements

1: Sterbe/Vermehrzeit

2: Touched this round

```
590     // Arrays for Border-Area
591     int *aBorderTop;
592     int *aBorderRight;
593     int *aBorderLeft;
594     int *aBorderBottom;
595
596     // +2 because Boder area is 2 Steps bigger then the normal area!!
597     aBorderTop = malloc(sizeof(int) * (iFieldSizeX+2));
598     aBorderBottom = malloc(sizeof(int) * (iFieldSizeX+2));
599     for (int x=0;x<(iFieldSizeX+2);x++)
600     {
601         aBorderTop[x] = 0;
602         aBorderBottom[x] = 0;
603     }
604
605     aBorderLeft = malloc(sizeof(int) * (iFieldSizeY+2));
606     aBorderRight = malloc(sizeof(int) * (iFieldSizeY+2));
607
608     for (int x=0;x<(iFieldSizeY+2);x++)
609     {
610         aBorderLeft[x] = 0;
611         aBorderRight[x] = 0;
612     }
```

Fish & Shark

MPI - Senden

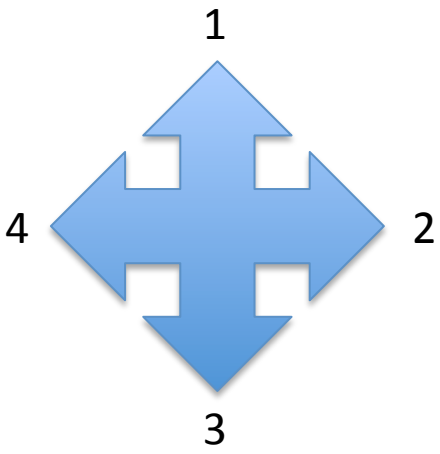
```
634         if (checkIfBorder(y, x)>0)
635         {
636             if (iDebugging == 2 && mpiIMe == iDebugProcess)
637                 printf("Border A1\n");
638
639             int temp = checkIfBorder(y, x);
640             mpiAData[0] = -2;
641             mpiAData[1] = -2;
642             mpiAData[2] = y;
643             mpiAData[3] = x;
644             mpiAData[4] = iMarkShark;
645
646             // Sent Data to the Process above
647             if ((temp == 1) || (temp == 6) || (temp == 7))
648             {
649                 MPI_Isend(mpiAData, 5, MPI_INT , mpiITop, 0, MPI_COMM_WORLD, &mpiRRequest);
650                 MPI_Wait(&mpiRRequest,&mpiStat);
651             }
652
653             if ((temp == 2) || (temp == 7) || (temp == 8))
654             {
655                 MPI_Isend(mpiAData, 5, MPI_INT , mpiIRight, 0, MPI_COMM_WORLD, &mpiRRequest);
656                 MPI_Wait(&mpiRRequest,&mpiStat);
657             }
658
659             if ((temp == 3) || (temp == 9) || (temp == 8))
660             {
661                 MPI_Isend(mpiAData, 5, MPI_INT , mpiIBottom, 0, MPI_COMM_WORLD, &mpiRRequest);
662                 MPI_Wait(&mpiRRequest,&mpiStat);
663             }
664
665             if ((temp == 4) || (temp == 6) || (temp == 9))
666             {
667                 MPI_Isend(mpiAData, 5, MPI_INT , mpiILeft, 0, MPI_COMM_WORLD, &mpiRRequest);
668                 MPI_Wait(&mpiRRequest,&mpiStat);
669             }
670         }
671
672     }
```

Datentransferformat

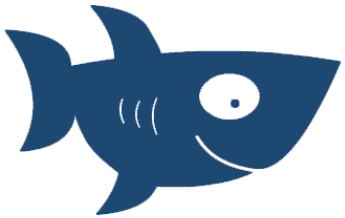
mpiAData[0-4]

0: Alte Y Position
1: Alte X Position
2: Neue Y Position
3: Neue X Position
4: Elementtyp

checkIfBorder(y,x)



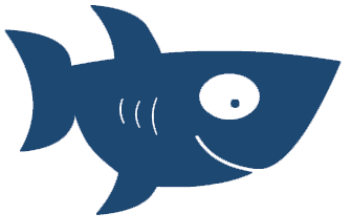
Fish & Shark



MPI - Empfangen

```
814                 // ------
815                 // MPI Recieve Stuff
816                 // ------
817
818                 // MPI: Check if another Process wants to transmit a Border-Movement
819                 MPI_Iprobe(mpiITop, 0, MPI_COMM_WORLD, &mpiIBorderTouched, &mpiStat);
820
821                 // If there is Data to recieve => Recieve it
822                 while (mpiIBorderTouched == 1) {
823                     if (iDebugging == 2 && mpiIMe == iDebugProcess)
824                         printf("Received from Top\n");
825
826                     // Recieve Stuff
827                     MPI_Irecv(mpiAData, 5, MPI_INT, mpiITop, 0, MPI_COMM_WORLD, &mpiRRequest);
828                     specialMovement(aField, aBorderTop, aBorderRight, aBorderBottom, aBorderLeft, mpiAData, 1, iIntervalCount);
829
830                     MPI_Iprobe(mpiITop, 0, MPI_COMM_WORLD, &mpiIBorderTouched, &mpiStat);
831                 }
832
833                 // MPI: Check if another Process wants to transmit a Border-Movement
834                 MPI_Iprobe(mpiIRight, 0, MPI_COMM_WORLD, &mpiIBorderTouched, &mpiStat);
835
836                 // If there is Data to recieve => Recieve it
837                 while (mpiIBorderTouched == 1) {
838
839                     if (iDebugging == 2 && mpiIMe == iDebugProcess)
840                         printf("Received from Right\n");
841
842                     // Recieve Stuff
843                     MPI_Irecv(mpiAData, 5, MPI_INT, mpiIRight, 0, MPI_COMM_WORLD, &mpiRRequest);
844                     specialMovement(aField, aBorderTop, aBorderRight, aBorderBottom, aBorderLeft, mpiAData, 2, iIntervalCount);
845
846                     MPI_Iprobe(mpiIRight, 0, MPI_COMM_WORLD, &mpiIBorderTouched, &mpiStat);
847                 }
848
849                 // MPI: Check if another Process wants to transmit a Border-Movement
850                 MPI_Iprobe(mpiIBottom, 0, MPI_COMM_WORLD, &mpiIBorderTouched, &mpiStat);
851
852                 // If there is Data to recieve => Recieve it
853                 while (mpiIBorderTouched == 1) {
854
855                     if (iDebugging == 2 && mpiIMe == iDebugProcess)
856                         printf("Received from Bottom\n");
857
858                     // Recieve Stuff
```

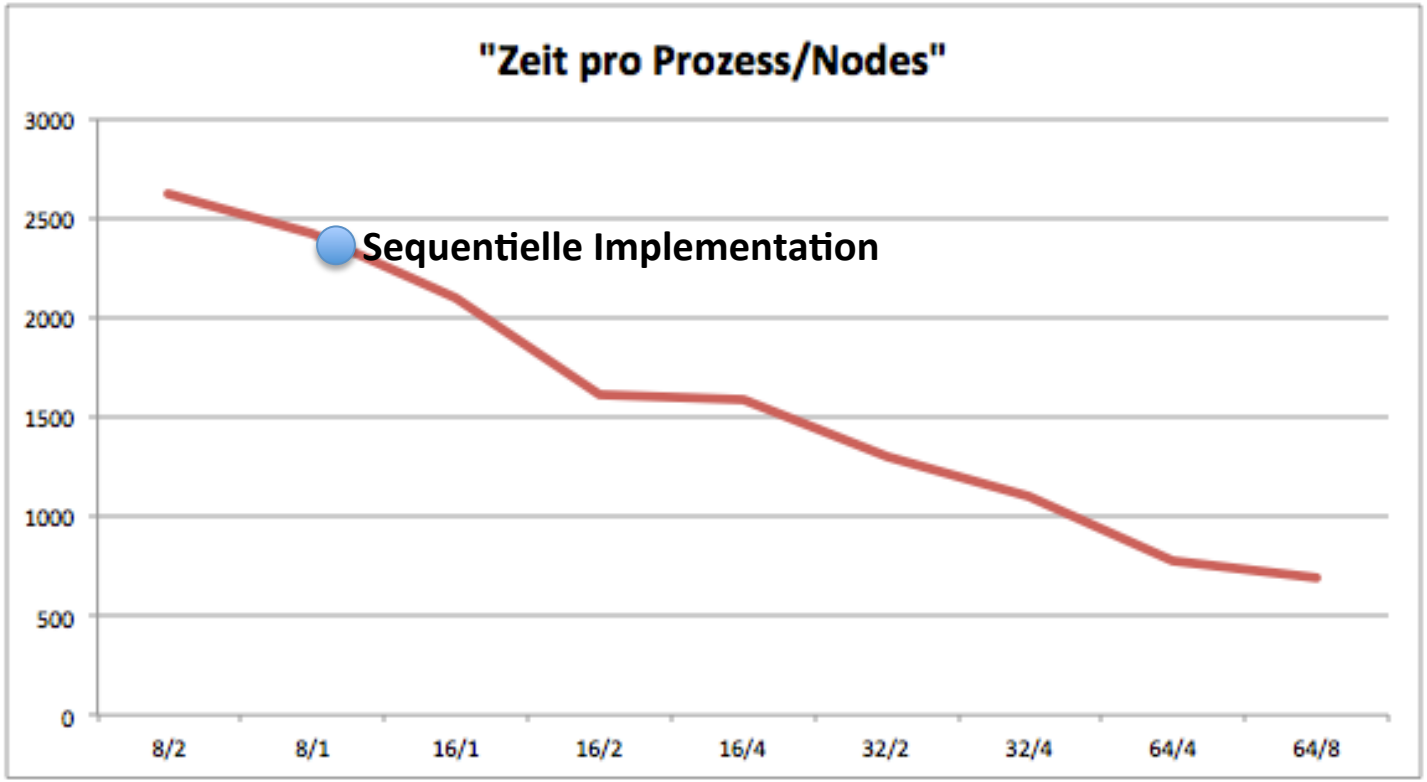
Fish & Shark



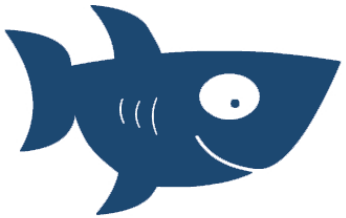
Verarbeiten

```
365 void specialMovement(int ***array, int *aBorderTop, int *aBorderRight, int *aBorderBottom, int *aBorderLeft, int *data, int from, int
        actual)
366 {
367
368     // Specification for from:
369     // 1 = Top
370     // 2 = Right
371     // 3 = Bottom
372     // 4 = Left
373
374     // Specification for Data Array:
375     // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
376     // !! Be Carful Position is from Process sending the Data !!
377     // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
378     // 0 = OLD-Y
379     // 1 = OLD-X
380     // 2 = NEW-Y
381     // 3 = NEW-X
382     // 4 = Type
383
384     if (iDebugging == 2)
385         printf("Special Movement: %d/%d/%d/%d/%d/%d\n", from, data[0], data[1], data[2], data[3], data[4]);
386
387     // *TOP*
388     if (from == 3)
389     {
390         // Border Area
391         if (data[2] == (iFieldSizeY-1))
392         {
393             // Check if Items has been in Border before!!
394             if (data[0] == (iFieldSizeY-1))
395             {
396                 aBorderTop[(data[1])] = 0;
397             }
398             aBorderTop[(data[3])] = data[4];
399         // Process Change
400         } else if (data[2] == iFieldSizeY)
401         {
402             if (data[0] == (iFieldSizeY-1))
403             {
404                 aBorderTop[(data[1])] = 0;
405             }
406             setValue(array, 0, data[3], data[4], actual);
407         } else if (data[2] == -2)
408         {
409             if (data[0] == (iFieldSizeY-1))
410             {
411                 aBorderTop[(data[1])] = 0;
412             }
413         }
```

Analyse



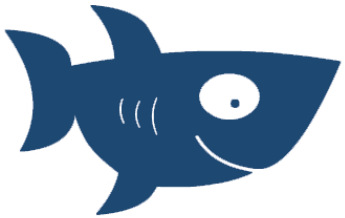
Parameter: 20.000 x 10.000 Spielfeld; 50 % Fische; 5 % Haie; 1 Plankton; 100 Bilder



Visualisierung

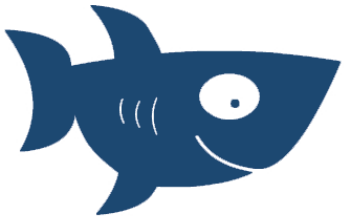
- Einlesen der generierten Dateien in alphabetischer Reihenfolge
- Erste Zeile enthält Zusatzinformationen

  -> Anzahl Prozesse, Größe der "Teilozeane“
- Danach: X-Koordinate – Element
- Y-Koordinate durch Zeilennummer
- Nachträglich Änderungen durch Prozesskommunikation
- http://bit.ly/TVghQu



Aufgetretene Probleme

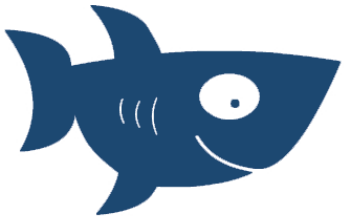
- Endlosschleifen
- Blockierendes Senden
- Nicht Blockierendes Senden ohne MPI_Wait
- Zu langes Suchen nach freier Position
- Signal 5: Maximale Dateigröße 1GB
- Signal 11: Zugriff auf nicht initialisierte Bereiche eines Arrays



Schnell leerendes Spielfeld

```
258 // Move all Values from one Sector to another
259 void moveValues(int ***array, int oldY, int oldX, int newY, int newX)
260 {
261     array[newY][newX][0] = array[oldY][oldX][0];
262     array[newY][newX][1] = array[oldY][oldX][1];
263     array[newY][newX][2] = 1;
264
265     resetSector(array, oldY, oldX);
266 }
```

```
258 // Move all Values from one Sector to another
259 void moveValues(int ***array, int oldY, int oldX, int newY, int newX)
260 {
261     array[newY][newX][0] = array[oldY][oldX][0];
262     array[newY][newX][1] = array[oldY][oldX][1];
263     array[newY][newX][2] = 1;
264
265     // Only Perform Move if real Movement
266     if ( (oldY != newY) || (oldX != newX))
267     {
268         resetSector(array, oldY, oldX);
269     }
270 }
```



Quellen

- Bild: http://www.harboarts.com/shirtdesigner
- http://de.wikipedia.org/wiki/Wator