Friedrich-Schiller-Universität Jena

8. Februar 2016, Sebastian Aschoff & Lucas Zoulkowski

Bruteforce Password Cracker

Parallelisierung mittels OpenMP und MPI

seit 1558

Inhaltsverzeichnis:

Problemkern I

- Finde eines Passwortes zu gegebenem Hash
→ Implementierung der Hashfunktionen

- Brute-Force oftmals einzige Möglichkeiten

- Problem: exponentielle Zahl von Passwörtern X^n

- Testen der Bruteforce Intervalle voneinander unabhängig

- Parallelisierung gut möglich

Problemkern II

- Parallelisierung von Hashfunktionen aufgrund definierter Eigenschaften mit OpenMp nicht möglich

→ daher wird die Berechnung der Bruteforce Intervalle parallelisiert

Vektorisierungspotenzial

→ Vektorisierung von Hashfunktionen

Hashfunktion	CRC32	MD5	SHA1	SHA256
Länge	8	32	40	64
Größe	32 Bit	128 Bit	160 Bit	256 Bit
Interne Block Verarbeitung in	32 Bit	4 mal 32 Bit Wörter	512 Bit Blöcke	512 Bit Blöcke

OpenMp vs. MPI Lösung

Erstellung der
Bruteforce
Intervalle mittels
Integer
Arrays

Beispiel:
(0 1 2 3 0)
entspricht
(a b c d a)

Arbeitspakete die
jeder
Prozess bekommt
besteht aus 2 Teilen

FstEntry

LstEntry

(1 0 0 0)

(2 0 0 0)

Feingranulare Berechnung der
Intervalle zur Arbeitspaketoptimierung
noch geplant, daher ist OpenMP
Implementierung effizienter

OpenMP Implementierung

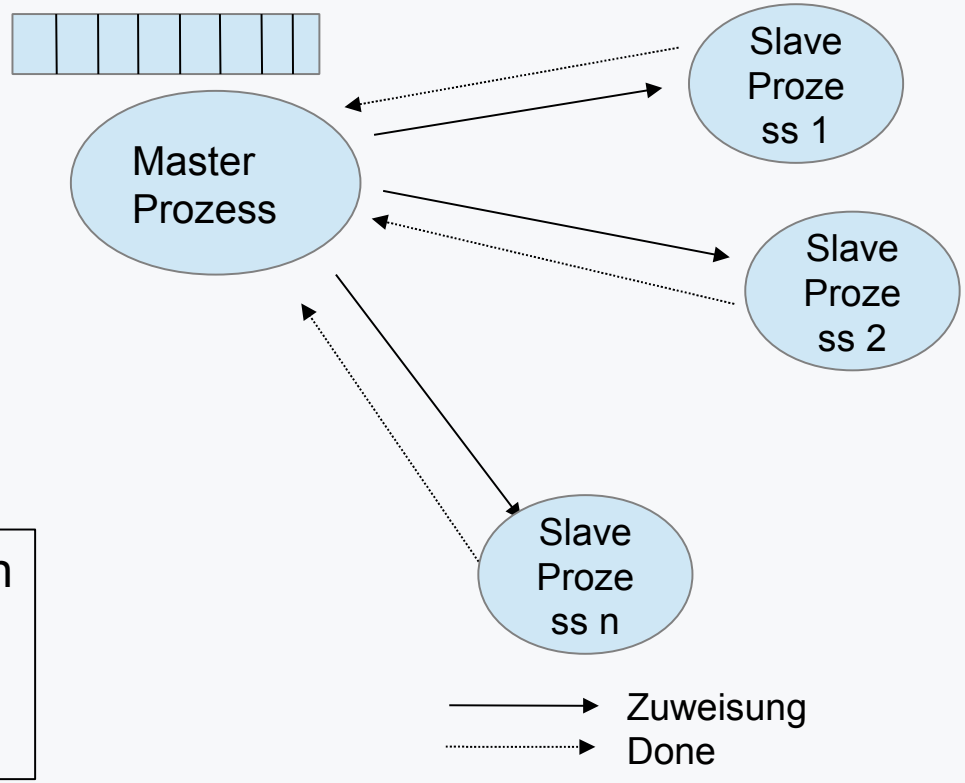
```
if (hashAlgo == 0 ) { //falls eingegebener String kein HashCode ist oder nicht unterstuetzt wird, dann return 0 und fertig
    MPI_Finalize();
    return 0;
}
else { //sonst starte Bruteforce Berechnung
  time( &StartTime ); //starte timer
    #pragma omp parallel shared( PasswordLength, BFOptionSize ) //Ein Thread pro Passwordlänge
    for( int PWLength = 1; PWLength < PasswordLength + 1; ++PWLength )
    {
        int FstEntry[PWLength], LstEntry[PWLength]; //Array-Größe für FirstEntry und LastEntry festlegen

        #pragma omp simd aligned(FstEntry)
        for( int j = 0; j < PWLength; ++j )
            FstEntry[j] = LstEntry[j] = 0; //Init mit Nullen


        #pragma omp for schedule( dynamic )
        for( int i = 0; i < *BFOptionSize; ++i ) //Loop, die für eine PWLänge alle möglichen Permutationen als Integer erzeugt
        {
            FstEntry[0] = i;
            LstEntry[0] = i + 1;
            bruteforceAttack( FstEntry, LstEntry, BFOption, BFOptionSize, PWLength ); //Rufe bruteforceAttack auf
        }
    }
  time( &EndTime ); //stoppe timer
    printf( "!!! Encrypted word not found !!! \n" );
    printf( "Elapsed time: %ld minutes %ld seconds\n\n", ( EndTime - StartTime ) / 60, ( EndTime - StartTime ) % 60 );
    MPI_Finalize();
    return 0;
}
```

MPI-Implementierung

Falls ein Prozess den Klartext findet, dann wird der Master benachrichtigt

Permutationen werden in Arbeitspakete eingeteilt, welche an die Prozesse verteilt werden

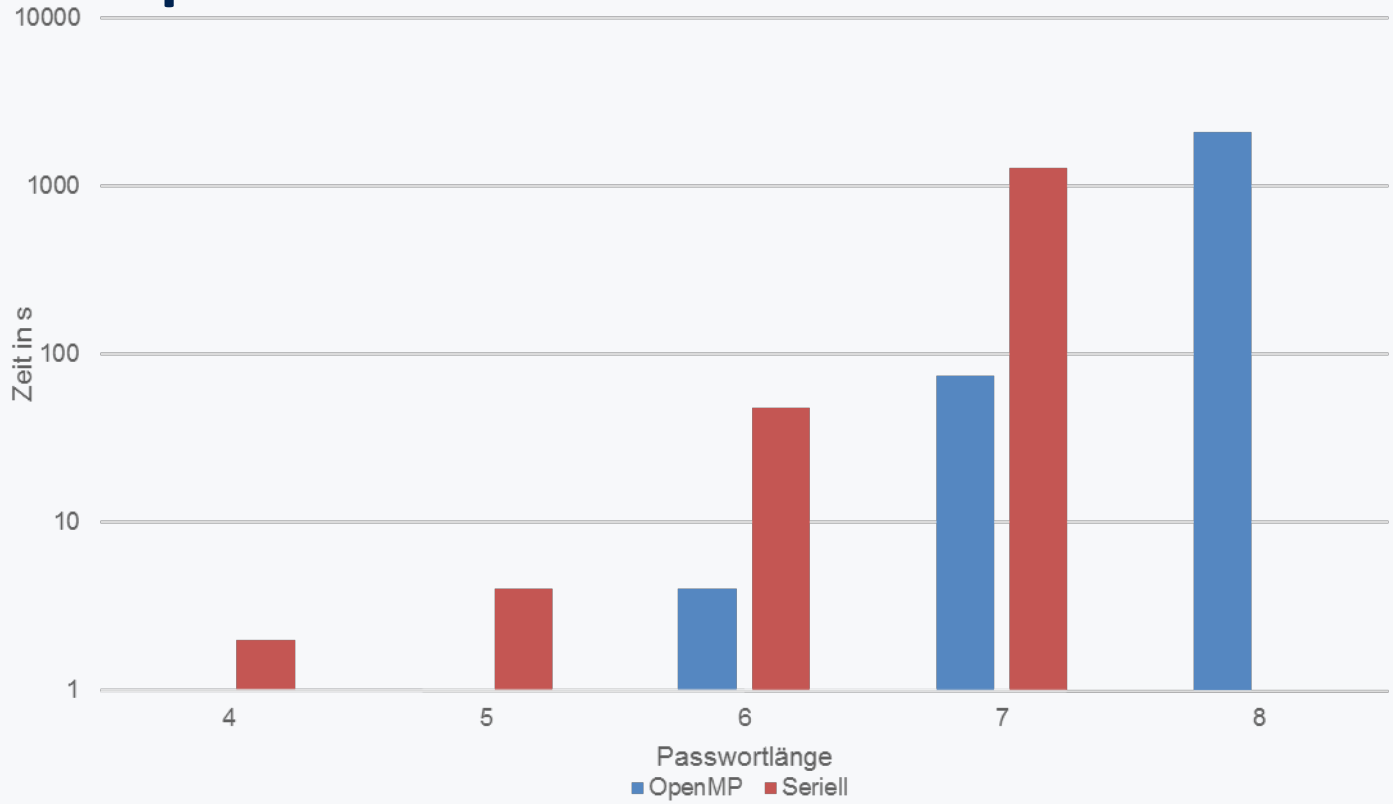
Beendet ein Prozess sein Arbeitspaket, dann wird ein neues Paket durch den Master zugewiesen



MPI-Implementierung
→ Master Thread Implementierung

```
// ARBEITSPAKETE WERDEN AN FREIE THREADS GESENDET
  while ((sendedRows < ( PasswordLength * (BFOptionSize)))&&(allDone==0)) {
      MPI_Request     findPWRequest;
      MPI_Request     allDoneRequest;
      MPI_Request     packageRequest;

      MPI_Irecv(&allDone, 1, MPI_INT, MPI_ANY_SOURCE, 42, MPI_COMM_WORLD, &findPWRequest);

      choosenThread = findIdleThread(activeThreads, size);

      if (choosenThread != -1) {

          MPI_Isend(&intervalArray[sendedRows][0],PasswordLength,MPI_INT,choosenThread,201,MPI_COMM_WORLD, &packageRequest);
          sendedRows = sendedRows + 1;
          setActiveThread(activeThreads, choosenThread);
          MPI_Isend(&allDone,1,MPI_INT,choosenThread,2,MPI_COMM_WORLD, &allDoneRequest);

      }
      else {

          MPI_Recv(&done,1,MPI_INT,MPI_ANY_SOURCE,100,MPI_COMM_WORLD,&status);

          MPI_Isend(&intervalArray[sendedRows][0],PasswordLength,MPI_INT,status.MPI_SOURCE,201,MPI_COMM_WORLD, &packageRequest);

          MPI_Isend(&allDone,1,MPI_INT,status.MPI_SOURCE,2,MPI_COMM_WORLD, &allDoneRequest);
          sendedRows = sendedRows + 1;
      }
  }


      allDone=1;
      MPI_Request     allDoneRequest;
      for (int l=1; l < size; ++l ) {
          MPI_Isend(&allDone,1,MPI_INT,l,2,MPI_COMM_WORLD, &allDoneRequest);
      }
```

Speed-Up

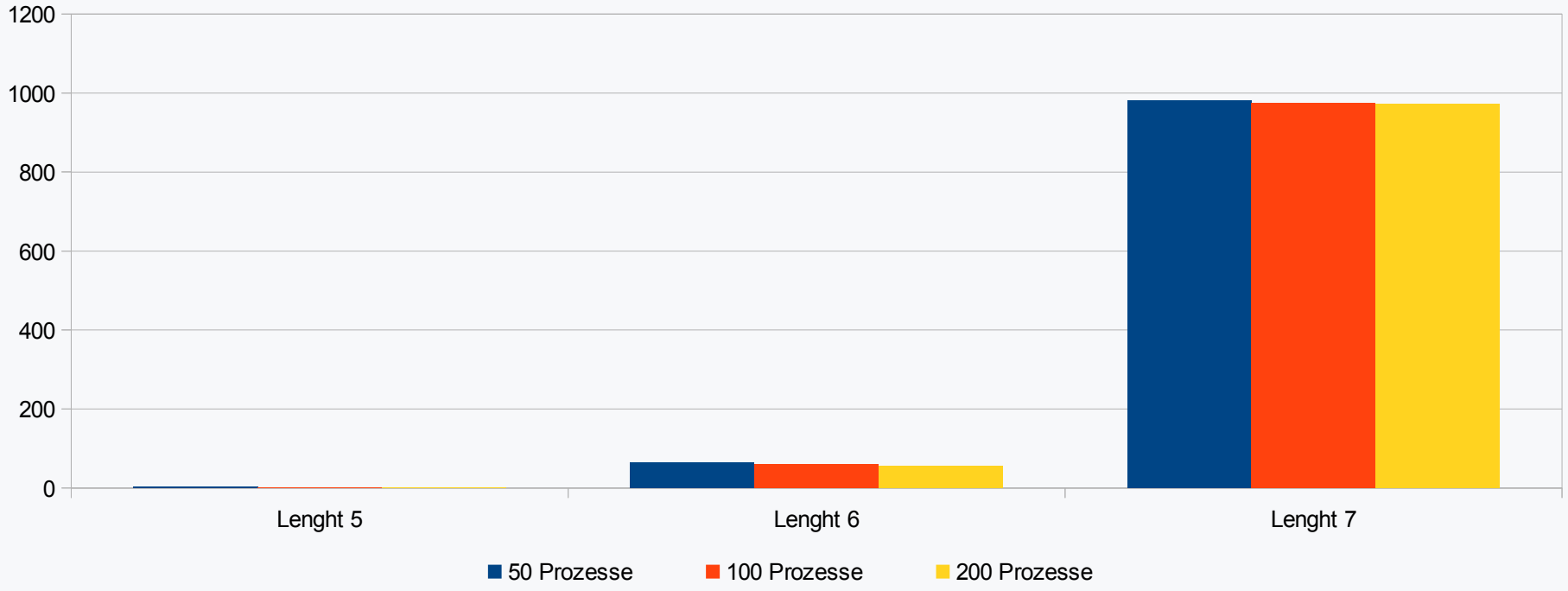


-Exponentielles Wachstum gut erkennbar

-Speed-Up steigt mit Passwortlänge

Speed-Up

- **Passworter sind SHA1 Hashcode mit zunehmender Länge und unterschiedlichen**



-Lesson learned I
-Parallelisiertes Knacken vieler Passwörter möglich

- exponentielles Wachstum macht Brute-Force jedoch schnell unattraktiv

- Konstruktion der for-Schleifen bei OpenMP Implementierung entscheidend

- Alternative Lösungen mit Blick auf umfangreichere Berechnungen

Lesson learned II

- Problematik der richtigen Verteilung von Arbeitspaketen (Datenstrukturen)
- Verteilung globaler Variablen an jeden Prozess
- Funktionalität der RequestQueue , als auch korrekte asynchrone Arbeitsweise schwierig zu grantieren

Vielen Dank für Ihre Aufmerksamkeit!

seit 1558

www.uni-jena.de