Peg-Solitaire

Florian Ehmke

29. März 2011

Gliederung

Einleitung

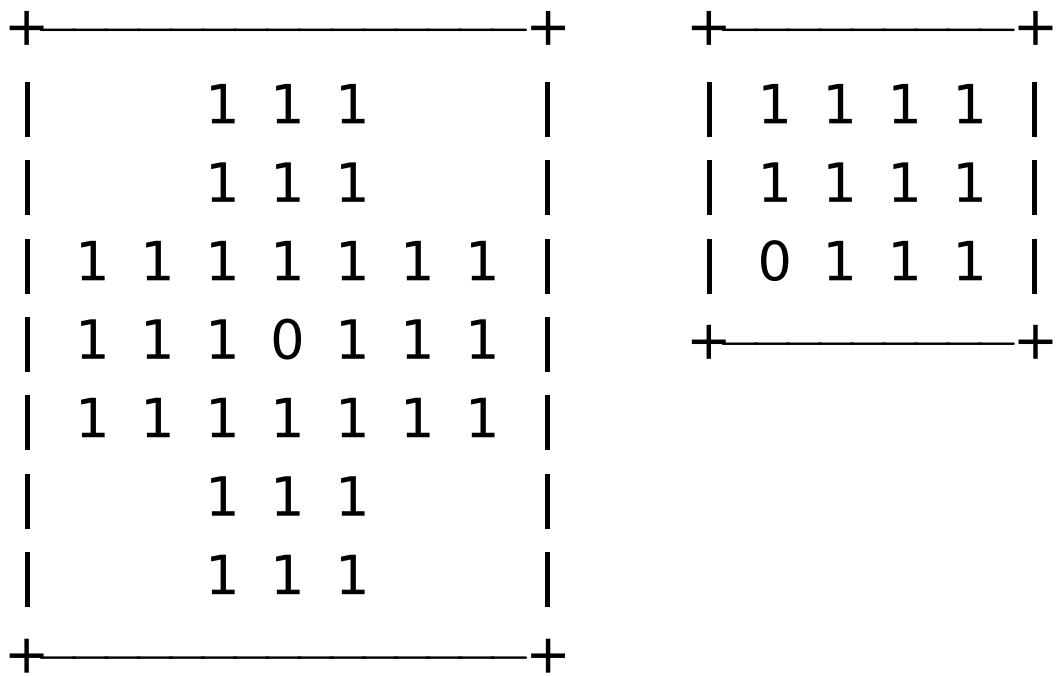
Das Spiel - Fakten Peg-33

- 33 Löcher, 32 Steine
- Gelöst wenn letzter Stein in Startposition
- 40.861.647.040.079.968 Lösungen
- 81.723.294.080.159.936 Sequenzen mit 31 Zügen
- 577,116,156,815,309,849,672 Sequenzen gesamt [1]

[1] Quelle: http://www.durangobill.com/Peg33.html

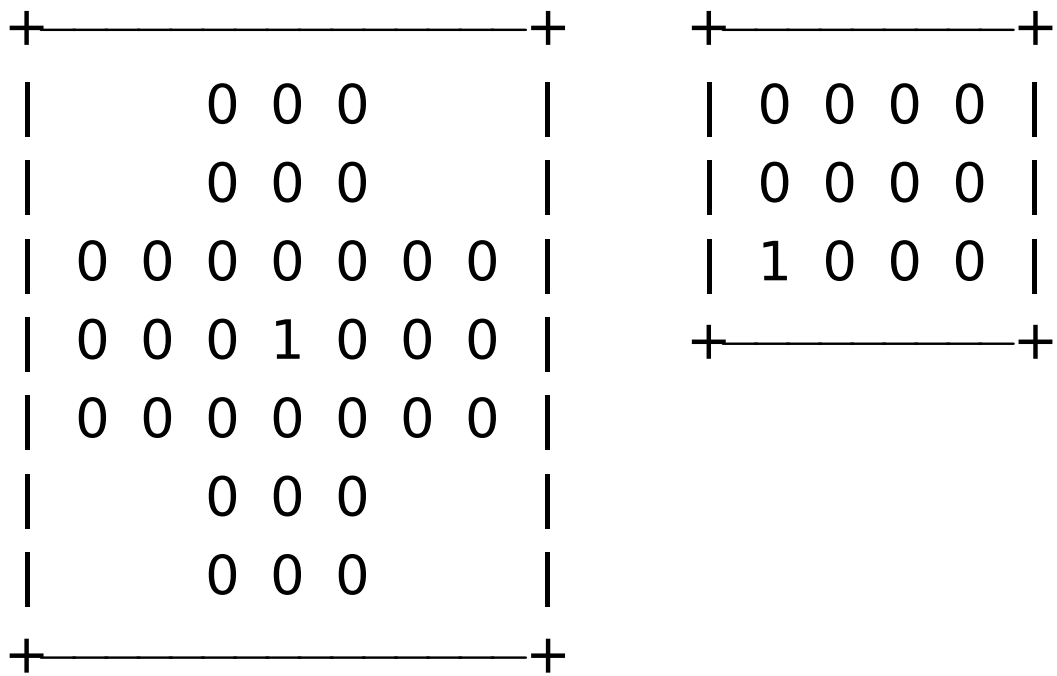
Einleitung

Das Spiel - Brett



Einleitung

Das Spiel - Brett gelöst



Einleitung

Das Ziel

40.861.647.040.079.968

Aufgabenstellung

- Sämtliche Lösungen für das Spiel zählen
  - Support für unterschiedliche Spielfelder
  - Auch Spiegelungen / Rotationen
- Lösungeswege nicht relevant
- Parallelisierung des Programms mittels MPI

Design und Implementierung

Probleme

- Spielzüge und Spielfeld effizient implementieren
- Tiefensuche nicht praktikabel
- Verwaltung der Spielfelder und Lösungen
- Zuordnung Board $\longleftrightarrow$ Prozess
- Sehr große Zahlen
- Lastungleichheit

Design und Implementierung

Design des Spielbretts

- Matrix als Spielfeld-Repräsentation ineffizient
  - Benötigt min. $7 \cdot 7 \cdot 8$ byte Speicher (bzw. $33 \cdot 8$)
- Ein 64bit integer genügt um ein Spielfeld zu codieren
  - Stein auf Position $\rightarrow$ Bit gesetzt
  - Position leer $\rightarrow$ Bit nicht gesetzt

- START `1111111111111110111111111111111 000..`
- ENDE `0000000000000001000000000000000 000..`

Design und Implementierung

Design eines Spielzuges

- Zug möglich in 4 verschiedene richtungen (N,S,W,O)
- Ein Feld wird übersprungen (und Stein dort entfernt)
- Beispiel:
  1. Vorher: Stein an Position (3,4) u. (3,5)
  2. (3,5) $\rightarrow$ (3,3)
  3. Nachher: Stein an Position (3,3)

Design und Implementierung

Design eines Spielzuges

- Spielzug besteht aus Check-Maske und Set-Maske
  - `Check: 00000011000000000000000000000000`
  - `Set:   00000011100000000000000000000000`
  - `Board: 11000111000100110001011110100 1010`
  - `Board & Set == Check?`
    - Ja: Zug möglich (Zug ausführen: `Board XOR Set`)
    - Nein: Zug nicht möglich

Design und Implementierung

Algorithmus - Tiefensuche

```
while (solutions left)
    for all moves
        if move is possible do move and continue
        else undo last move and try another one
```

Design und Implementierung

Algorithmus - Dynamische Programmierung

- Problem: Spielfelder mehrfach betrachtet
- Wenn ein Spielfeld das zweite mal auftritt Zähler erhöhen
  - $\rightarrow$ Synchronisation nötig
- Prozess bekommt Teilmenge der Spielfelder zugeordnet
  - Dynamische Programmierung
  - Breitensuche

Design und Implementierung

Algorithmus - Breitensuche

```
while (pegs on the board)
    for all boards assigned to this rank
        do all moves and store new boards
    for all new boards
        assign board to corresponding rank
    exchange boards with all ranks
```

Design und Implementierung

Algorithmus - Board $\rightarrow$ Prozess Zuweisung

- Eindeutige Zuordnung Board $\longleftrightarrow$ Prozess nötig
- Programm nur mit 2^x Prozessen startbar
  - Prozesse: 16
  - Rank: 1100
  - Board: 1100010100010011000100000001001010
  - Board: 1100011110101001110010111101011010
  - Board: 1101011100001001100010110001101110

Design und Implementierung

Algorithmus - Hashtables

- GLib Hashtables[1] für Boardverwaltung
- Jedes neue Board wird im Hashtable gespeichert (Key), wenn schon vorhanden wird Lösungszähler (Value) erhöht
- Daten werden zum verschicken in Arrays kopiert

```
while (pegs on the board)
     for all boards assigned to this rank
          do all moves and store new boards
     for all new boards
          assign board to corresponding rank
     exchange boards with all ranks
```

[1] http://library.gnome.org/devel/glib/2.24/glib-Hash-Tables.html

Design und Implementierung

Algorithmus - Kommunikation

1. Größe der Hashtables austauschen (`ISend` & `IRecv`)
2. Speicher für Send & Receive buffer allozieren
3. Send buffer füllen (Key & Value aus Hashtable kopieren)
4. Key & Value austauschen (2x `ISend` & `IRecv`)
5. Neuen Hashtable konstruieren

Ergebnisse

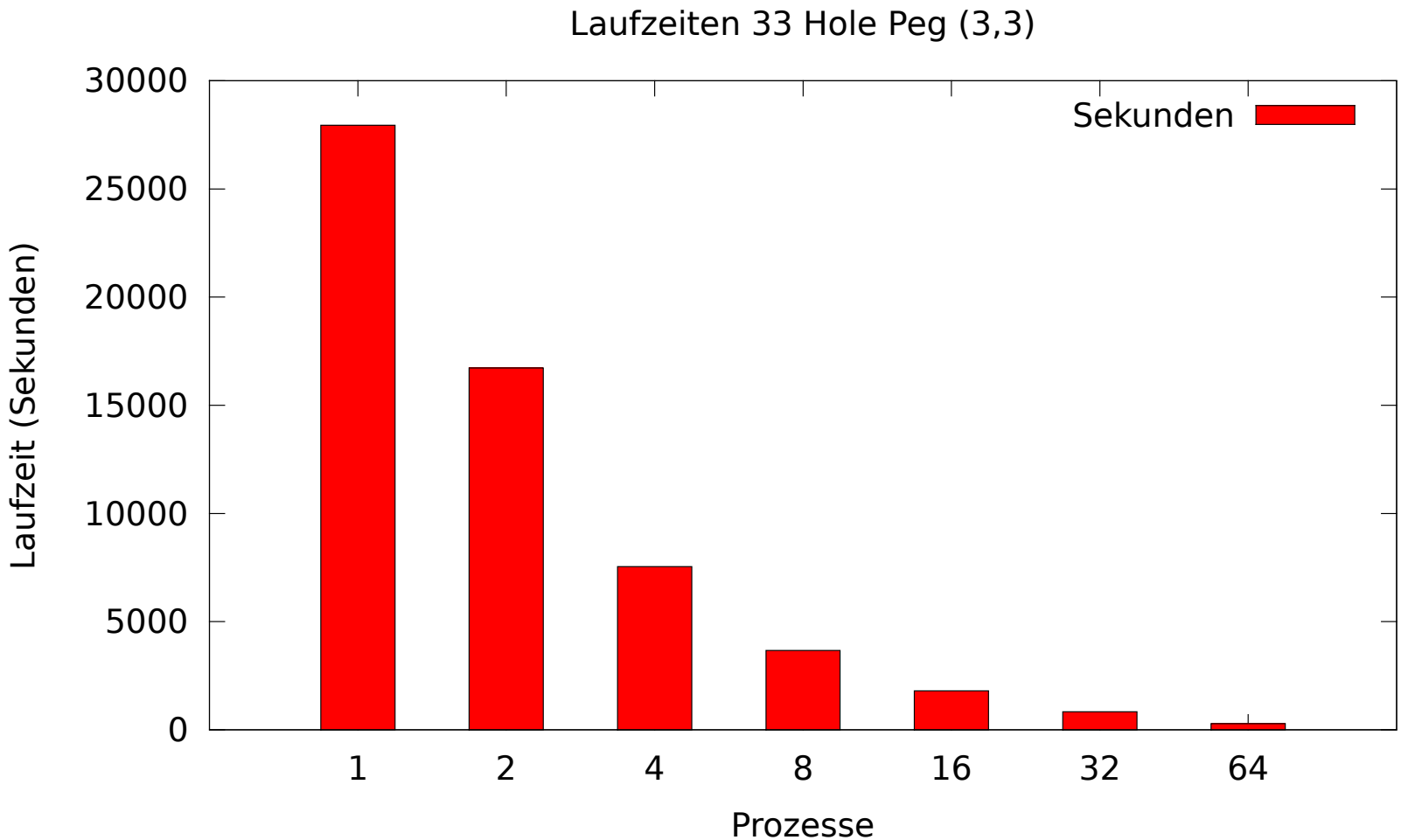
Algorithmus - Anzahl Lösungen

- Anzahl Lösungen gefunden für 33 Hole Peg mit der Startposition (3,3) = 4086164704007996 8
- Ergebnis ist korrekt [1] [2] [3]

[1] Quelle [1]: http://www.durangobill.com/Peg33.html
[2] Quelle [2]: http://home.iitk.ac.in/ sunithb/reports.html
[3] Quelle [3]: http://dauphinelle.free.fr/solitaire/index.php?page=solisolu

Ergebnisse

Laufzeit



Ergebnisse

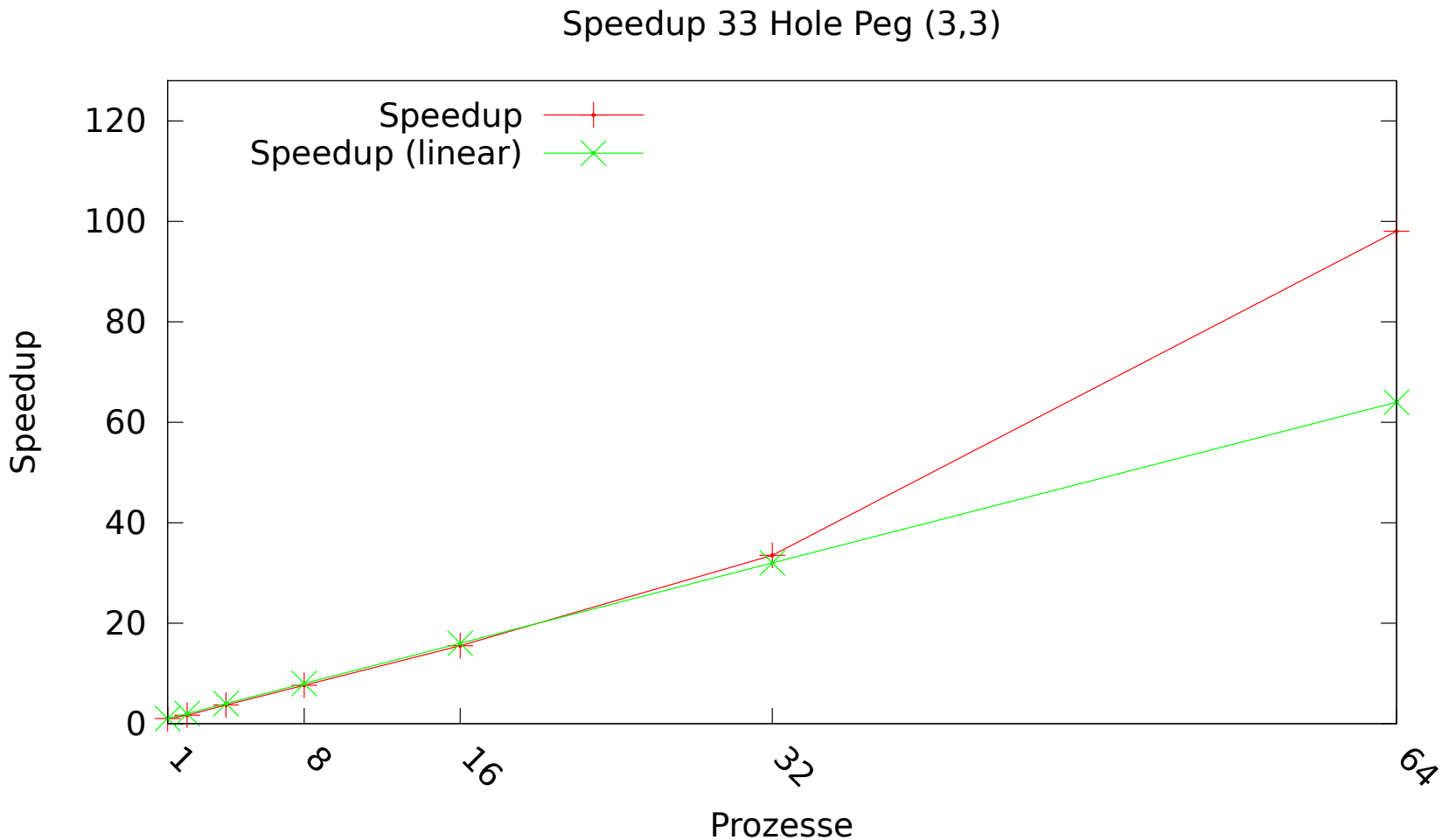
Lösungen / Sekunde

Prozesse	Lösungen / Sekunde
1 (Tiefensuche)	361
1	1462478419473
2	2442563634412
4	5415725253821
8	11149153353364
16	22688310405374
32	48994780623597
64	143374200140631

→ Tiefensuche, 1 Prozess bei 361 Lösungen pro Sekunde benötigt für 4086164704007996 8 Lösungen 31441710557 Stunden (ca. 3589236 Jahre).

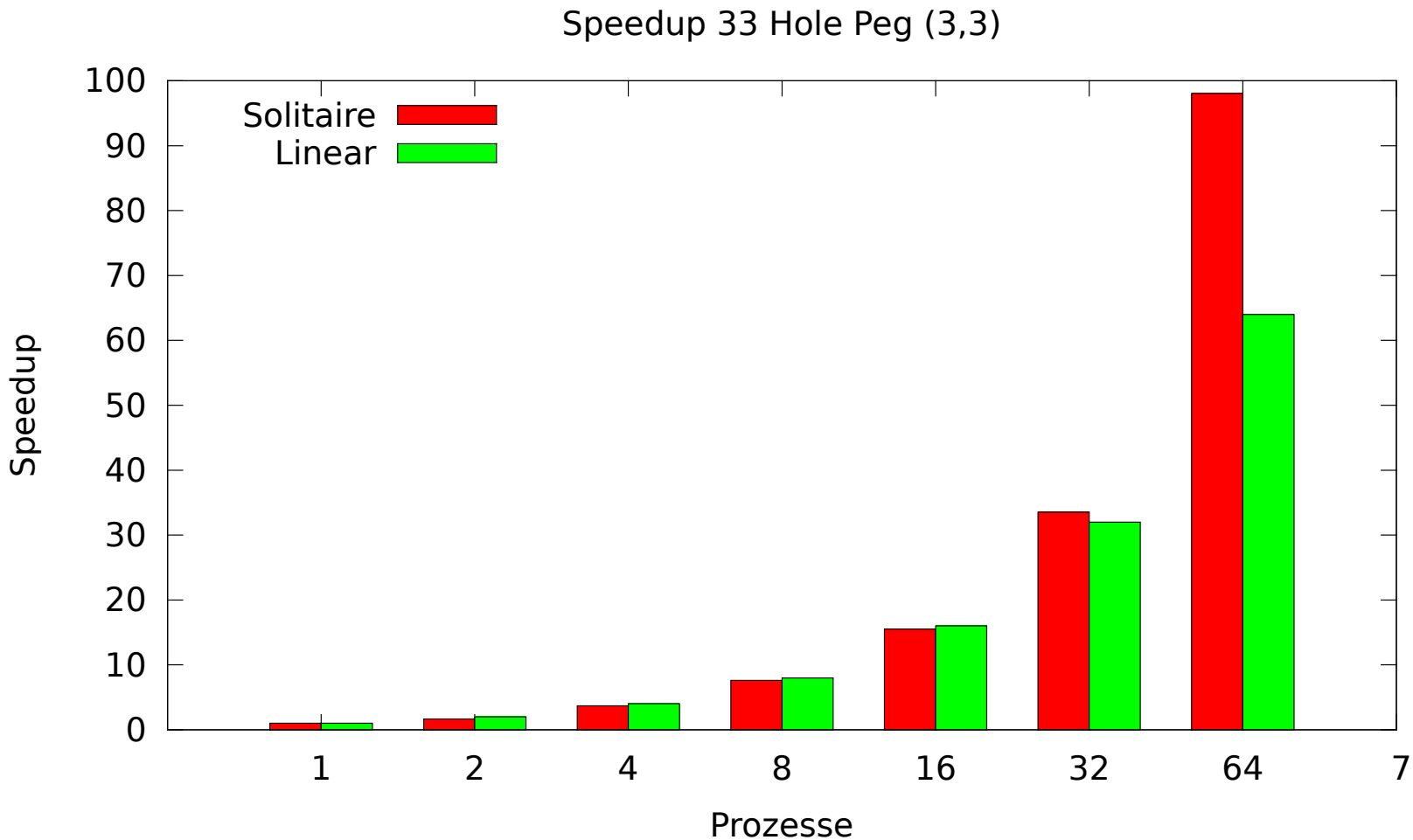
Ergebnisse

Speedup



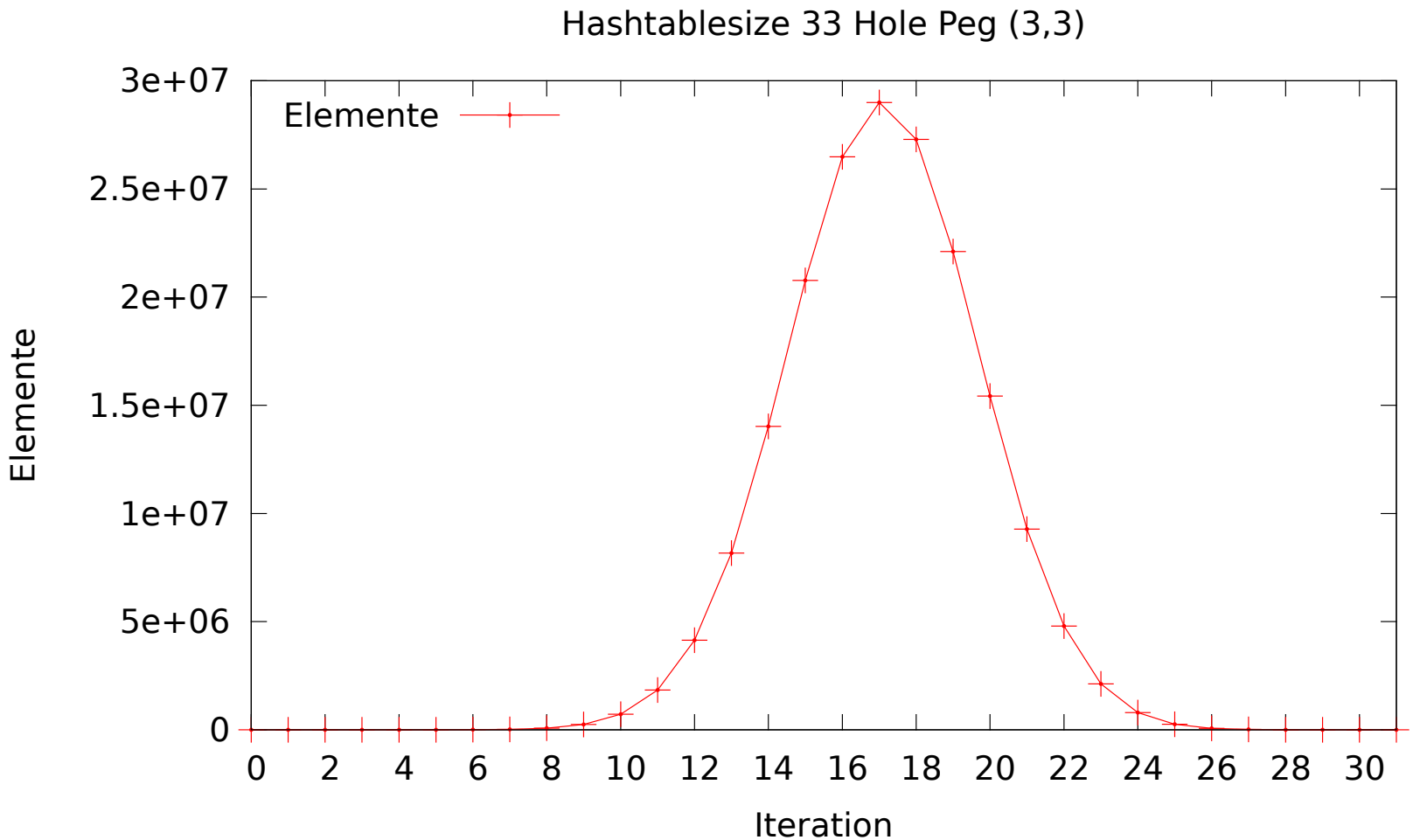
Ergebnisse

Speedup



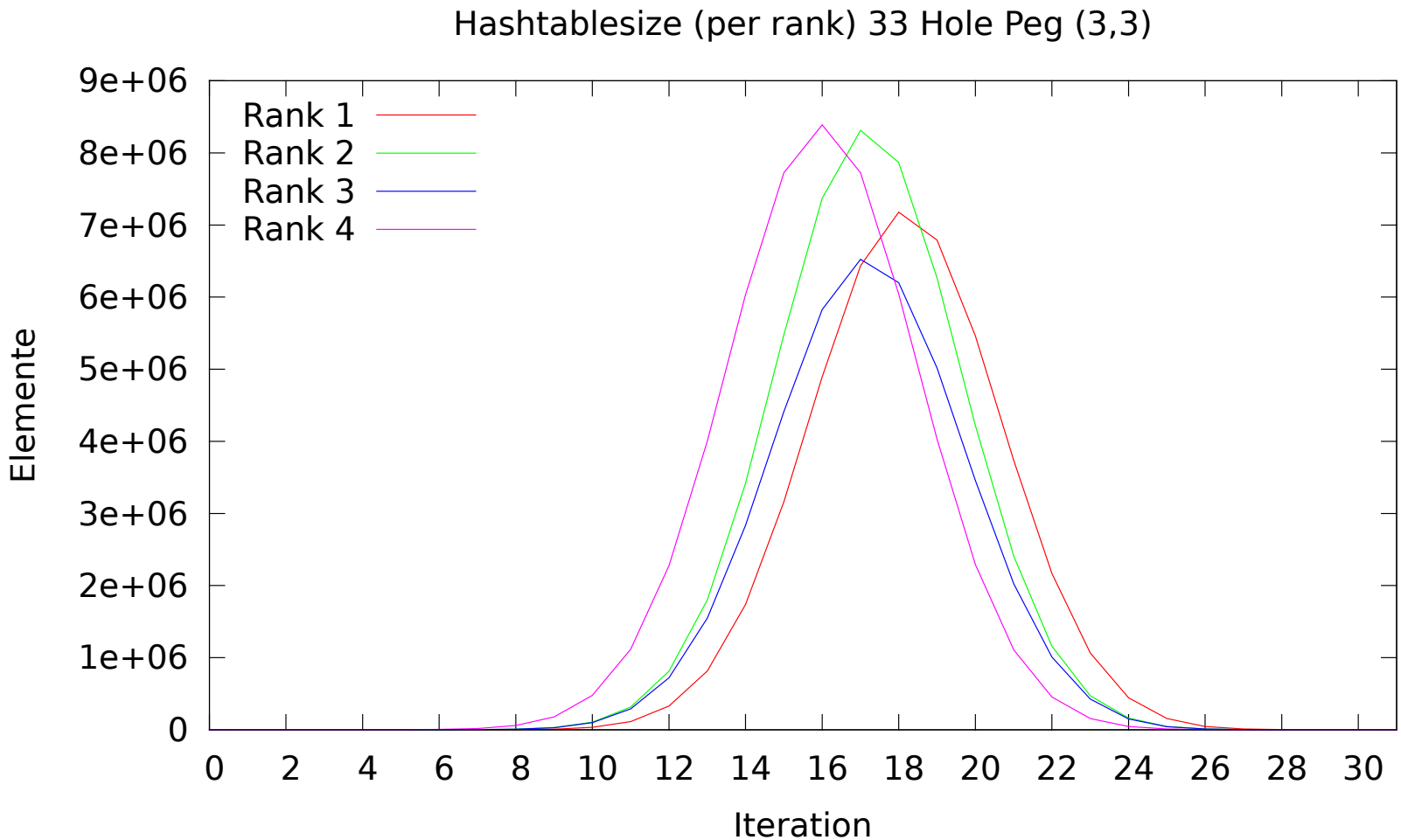
Ergebnisse

Größe der Hashtables



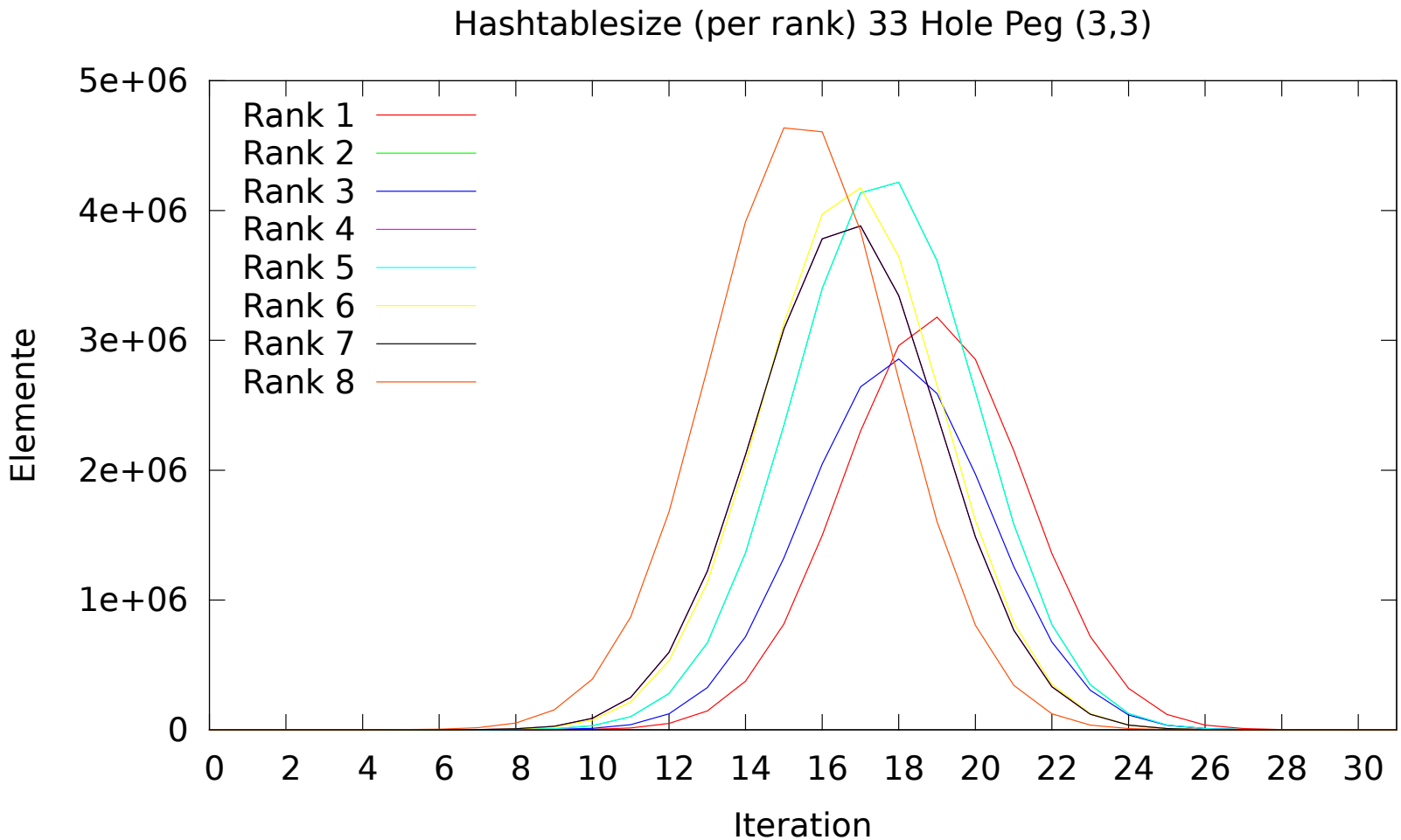
Ergebnisse

Größe der Hashtables - Verteilung



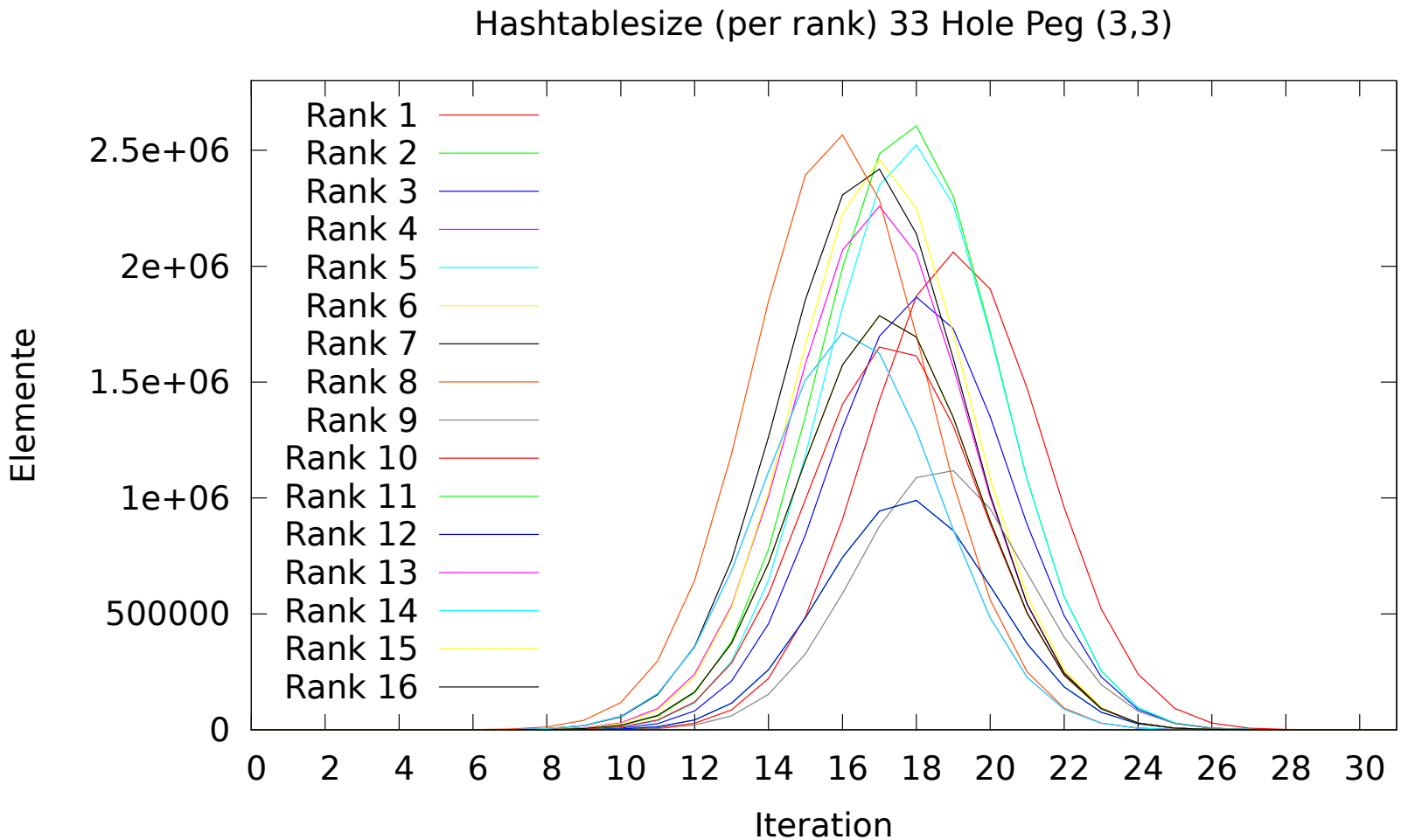
Ergebnisse

Größe der Hashtables - Verteilung



Ergebnisse

Größe der Hashtables - Verteilung



Ergebnisse

Tracing

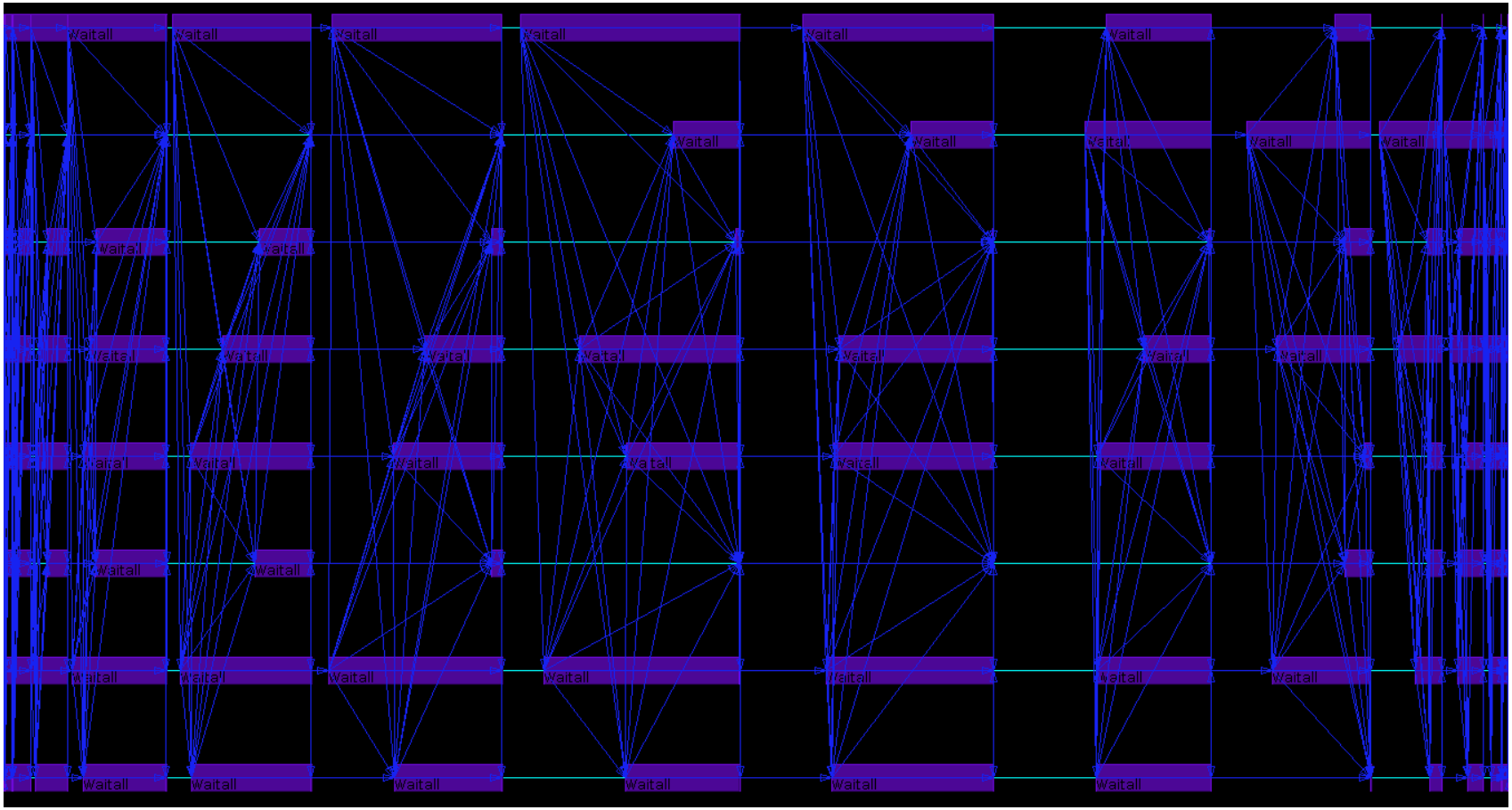


Abbildung: 33 Hole Peg (3,3) mit 8 Prozessen (1 pro Knoten)

Probleme / Todo

- Peg-33 mit Startposition (2,4)
  - `138,409,681,956,904,365,268` Lösungen[1]
  - `18,446,744,073,709,551,615 = maxValue(uint64)`
- Kommunikation optimieren (Lastungleichheit)
  - Keine statische Spielfeld-Zuweisung.
  - Nicht nach jeder Iteration synchronisieren

[1] Quelle: http://www.durangobill.com/Peg33.html