CSE 633: Parallel Algorithms

Fall 2012

Parallelized Hash Collision Attacking

Benedikt Budig

Course Instructor:
Russ Miller

The Topic in a Nutshell

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$,

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$,

that is, if h maps the two strings to the same hash value.

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$,

that is, if h maps the two strings to the same hash value.

$\longrightarrow$ find a string y such that $h(y)$ collides with given $h(x)$

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$,

that is, if h maps the two strings to the same hash value.

$\longrightarrow$ find a string y such that $h(y)$ collides with given $h(x)$

Reason to do that?

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$,

that is, if h maps the two strings to the same hash value.

$\longrightarrow$ find a string y such that $h(y)$ collides with given $h(x)$

Reason to do that: Cryptographic Applications

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$,

that is, if h maps the two strings to the same hash value.

$\longrightarrow$ find a string y such that $h(y)$ collides with given $h(x)$

Reason to do that: Cryptographic Applications

- secure storage of passwords

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$, that is, if h maps the two strings to the same hash value.

$\longrightarrow$ find a string y such that $h(y)$ collides with given $h(x)$

Reason to do that: Cryptographic Applications

- secure storage of passwords
- digital signature schemes

image source: shutterstock

The Topic in a Nutshell

A *hash function* is a total function $h : \{0,1\}^* \to \{0,1\}^n$ that maps arbitrarily long strings to strings of a fixed length n.

Project Goal: Find Hash Collisions for given Hash

A *hash collision* occurs for two strings x, y if $h(x) = h(y)$, that is, if h maps the two strings to the same hash value.

$\longrightarrow$ find a string y such that $h(y)$ collides with given $h(x)$

Reason to do that: Cryptographic Applications

- secure storage of passwords
- digital signature schemes

We focus on

$\longrightarrow$ MD5

image source: shutterstock

Parallel Approach

Parallel Approach

Input:

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output:

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach

1. based on their ID, m parallel processes take a subset of the possible strings $\{0, 1\}^{\leq n}$

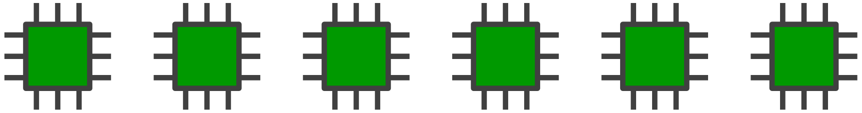
Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach

1. based on their ID, m parallel processes take a subset of the possible strings $\{0,1\}^{\leq n}$



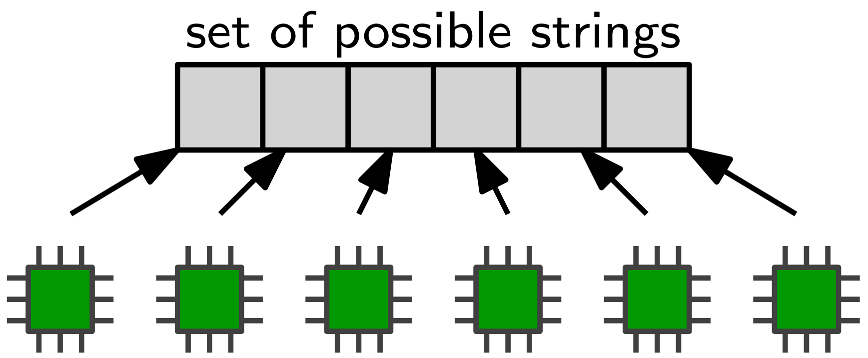
Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach

1. based on their ID, m parallel processes take a subset of the possible strings $\{0,1\}^{\leq n}$



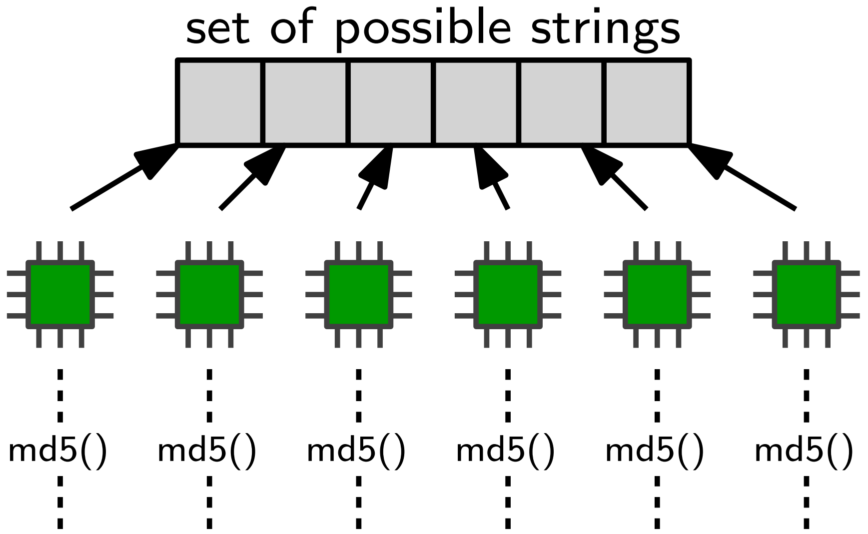
Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach

1. based on their ID, m parallel processes take a subset of the possible strings $\{0,1\}^{\leq n}$
2. each process calculates hashes of the strings assigned to it

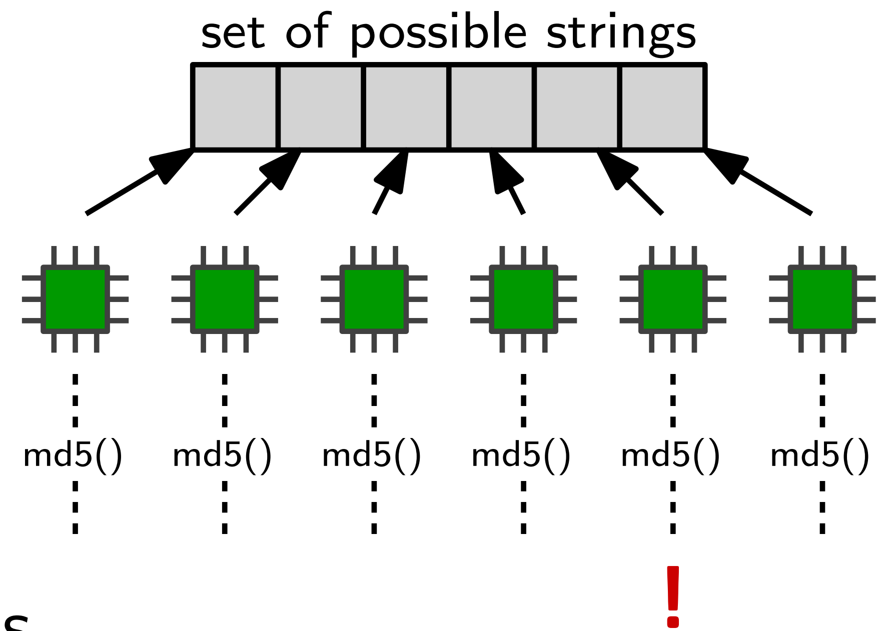


Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach



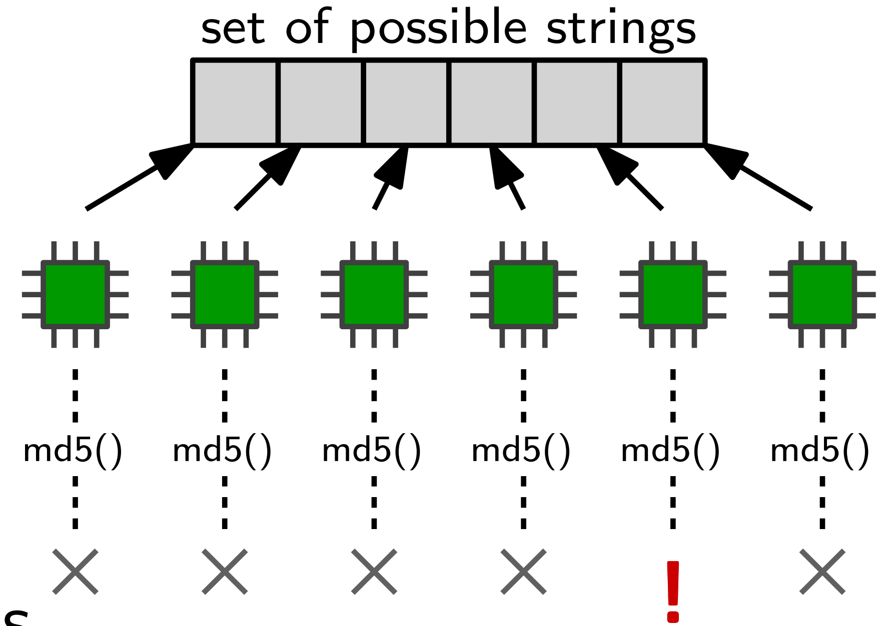
1. based on their ID, m parallel processes take a subset of the possible strings $\{0,1\}^{\leq n}$
2. each process calculates hashes of the strings assigned to it
3. as soon as one process calculates a hash equal to the input hash, all processes terminate

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach



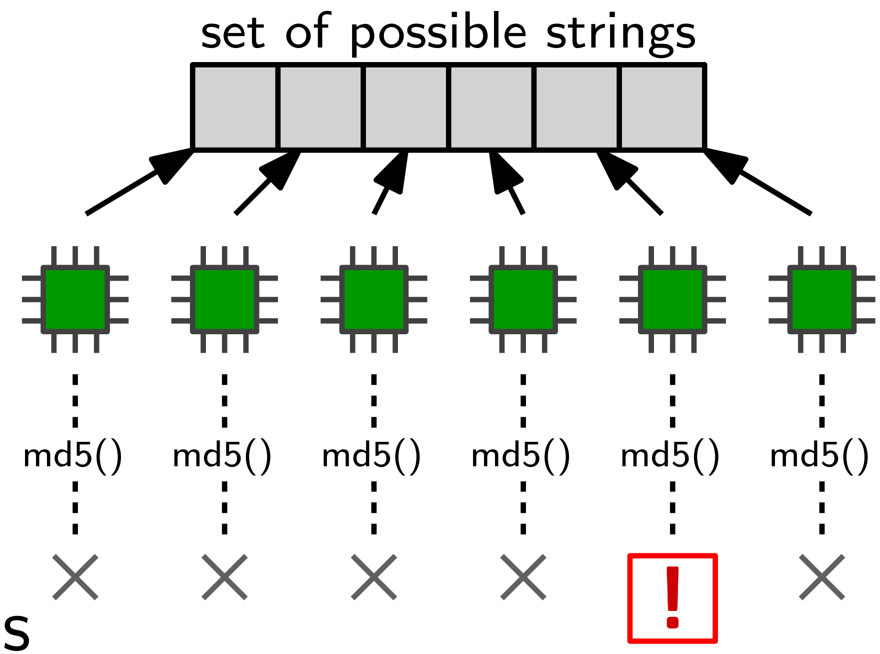
1. based on their ID, m parallel processes take a subset of the possible strings $\{0,1\}^{\leq n}$
2. each process calculates hashes of the strings assigned to it
3. as soon as one process calculates a hash equal to the input hash, all processes terminate

Parallel Approach

Input: hash md5(x) of unknown string x of length $|x| \leq n$

Output: string y such that md5(y) $=$ md5(x) and $|y| \leq n$

Parallel approach



1. based on their ID, m parallel processes take a subset of the possible strings $\{0,1\}^{\leq n}$
2. each process calculates hashes of the strings assigned to it
3. as soon as one process calculates a hash equal to the input hash, all processes terminate
4. return the colliding string

Technical Realization

Technical Realization

Hardware

image source: CCR website

Technical Realization

Hardware

- use of CPUs

image source: CCR website

Technical Realization

Hardware

- use of CPUs
- use of a multi-core system

image source: CCR website

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

- implementation using the C++ programming language

Technical Realization


image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

- implementation using the C++ programming language
- one implementation using OpenMP

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

- implementation using the C++ programming language
- one implementation using OpenMP
- and another version using MPI

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

- implementation using the C++ programming language
- one implementation using OpenMP
- and another version using MPI

Future Work

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

- implementation using the C++ programming language
- one implementation using OpenMP
- and another version using MPI

Future Work

- tweaks for MPI and OpenMPI

Technical Realization

image source: CCR website

Hardware

- use of CPUs
- use of a multi-core system
- Infiniband network
- tests on CCR machines with 12 cores and 32 cores

Software and Implementation

- implementation using the C++ programming language
- one implementation using OpenMP
- and another version using MPI

Future Work

- tweaks for MPI and OpenMPI
- implementation using CUDA

Benchmarks

image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz

image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$

image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$

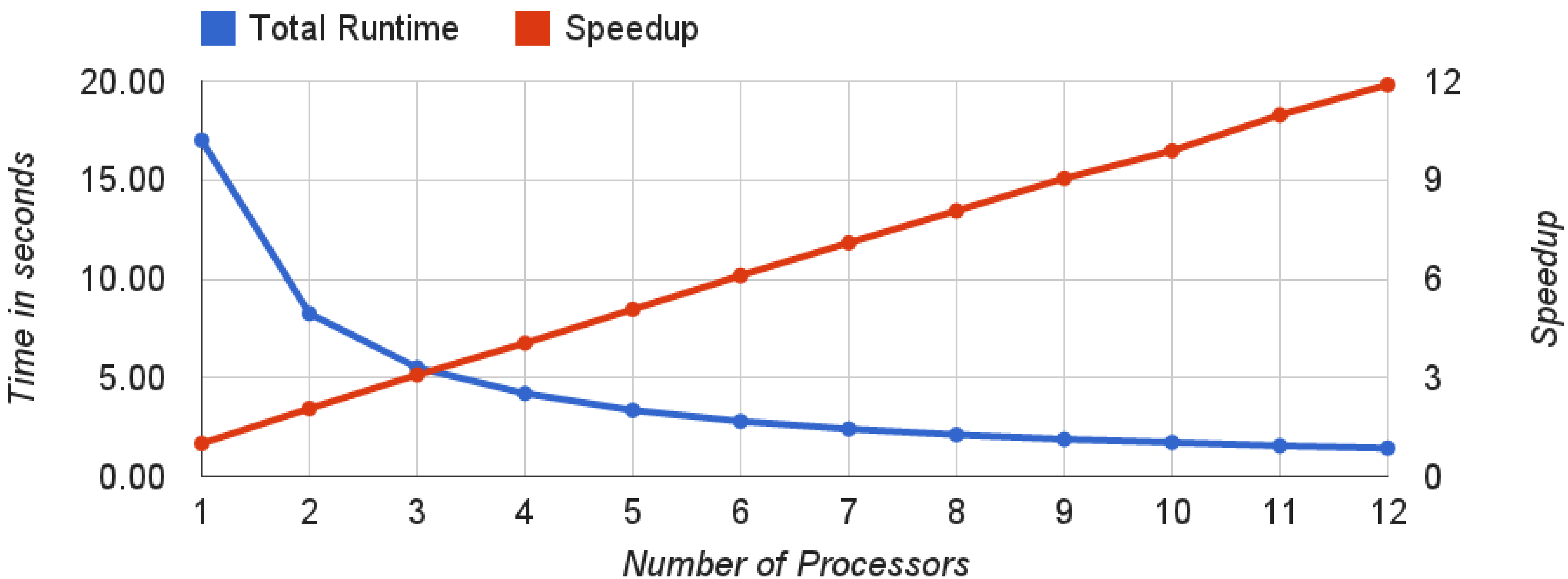
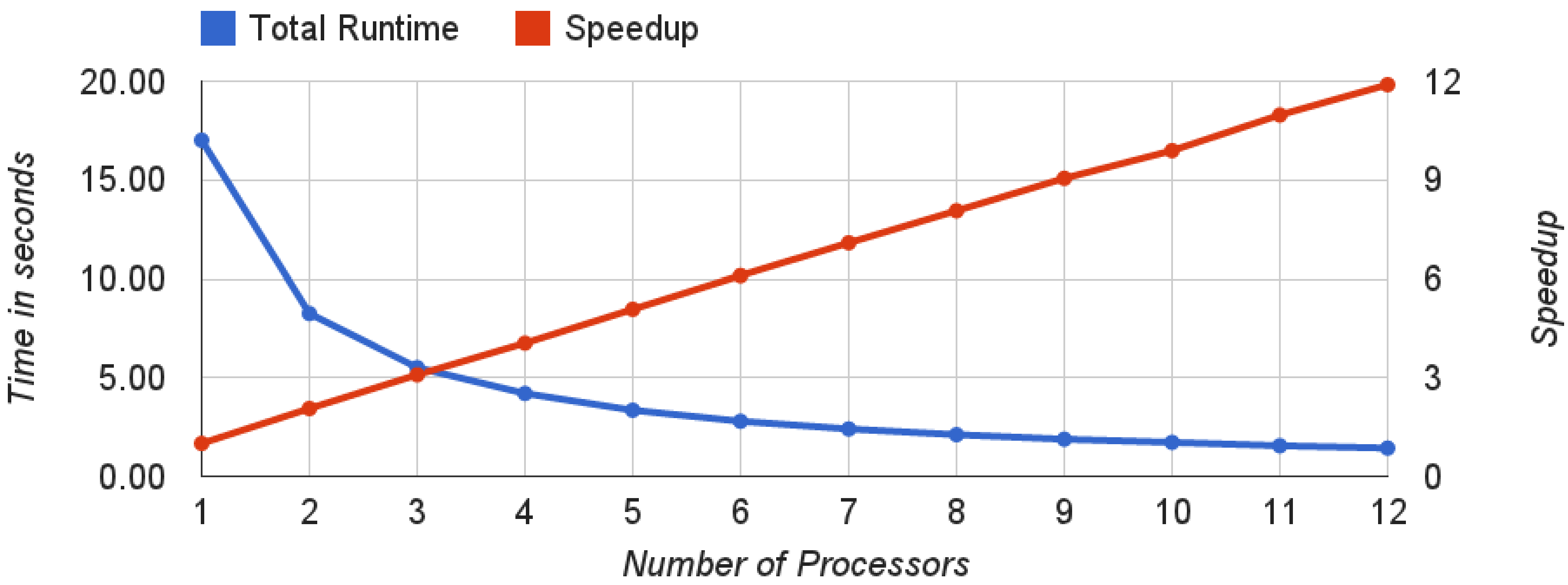


image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



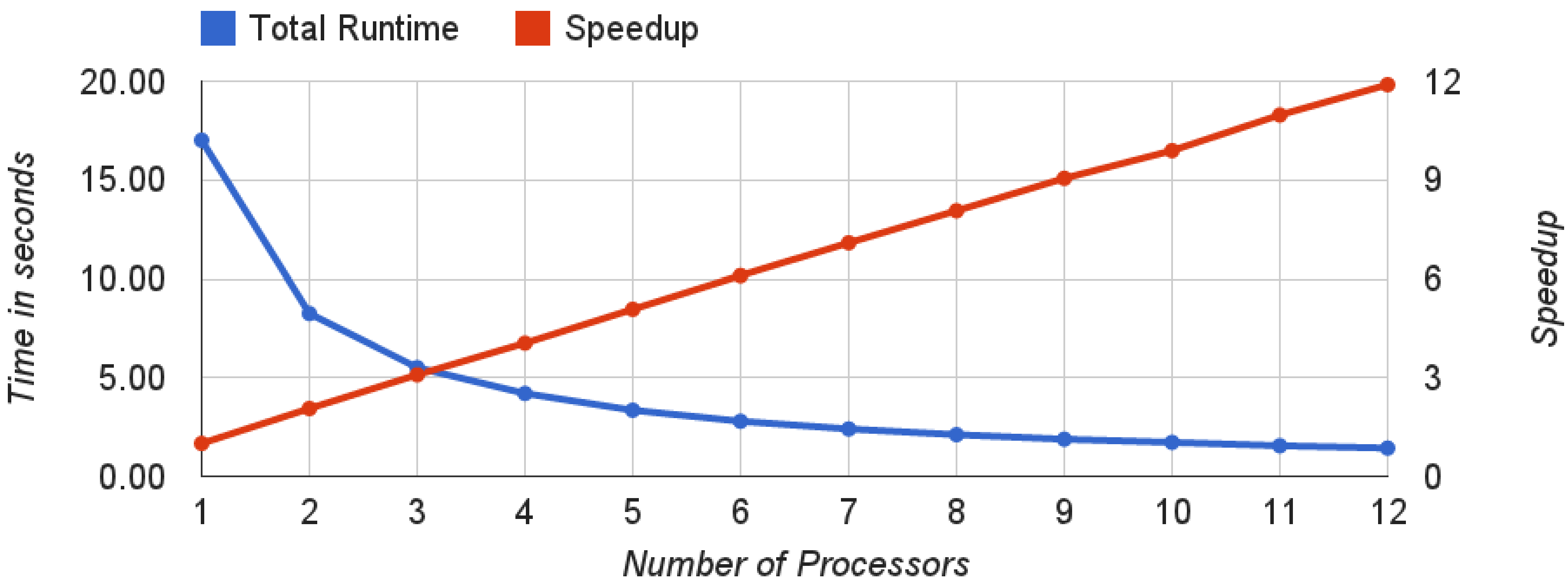
- 1.43 seconds to find collision

image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



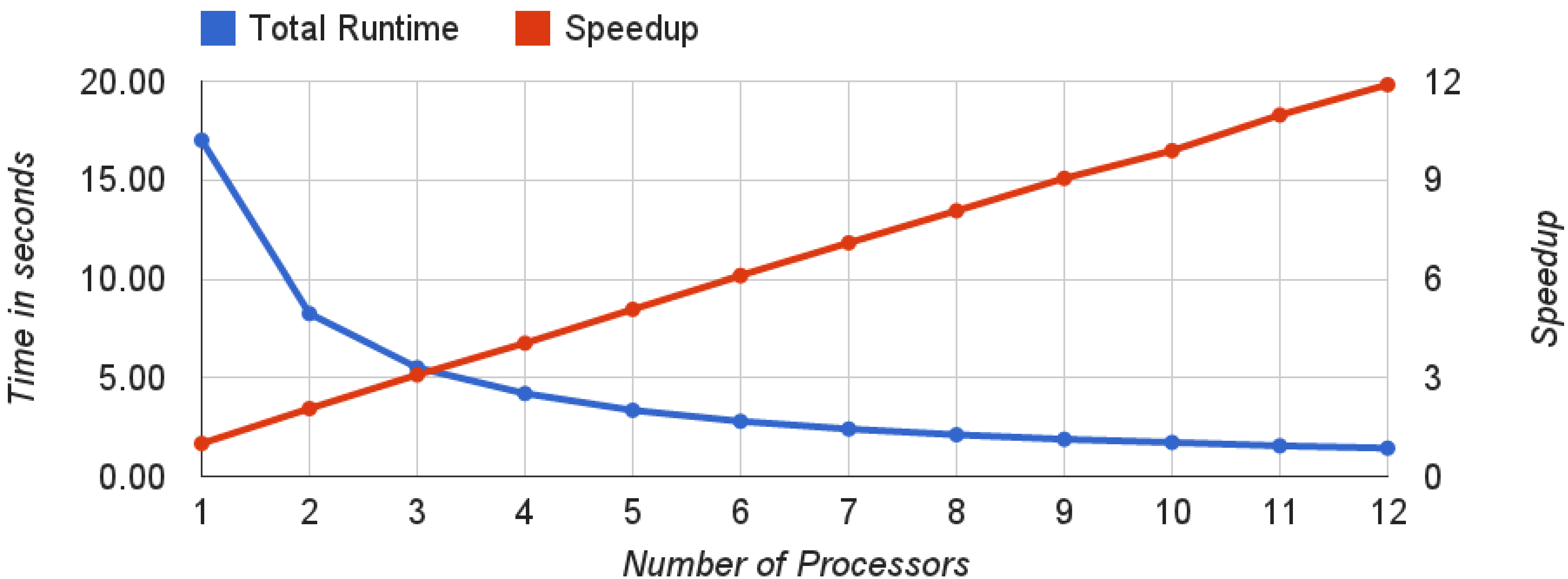
- 1.43 seconds to find collision
- approx. 4 million strings tried

image source: autoanything

Benchmarks

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



- 1.43 seconds to find collision
- approx. 4 million strings tried

→ linear speedup

image source: autoanything

Benchmarks

image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz

image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$

image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{32}$

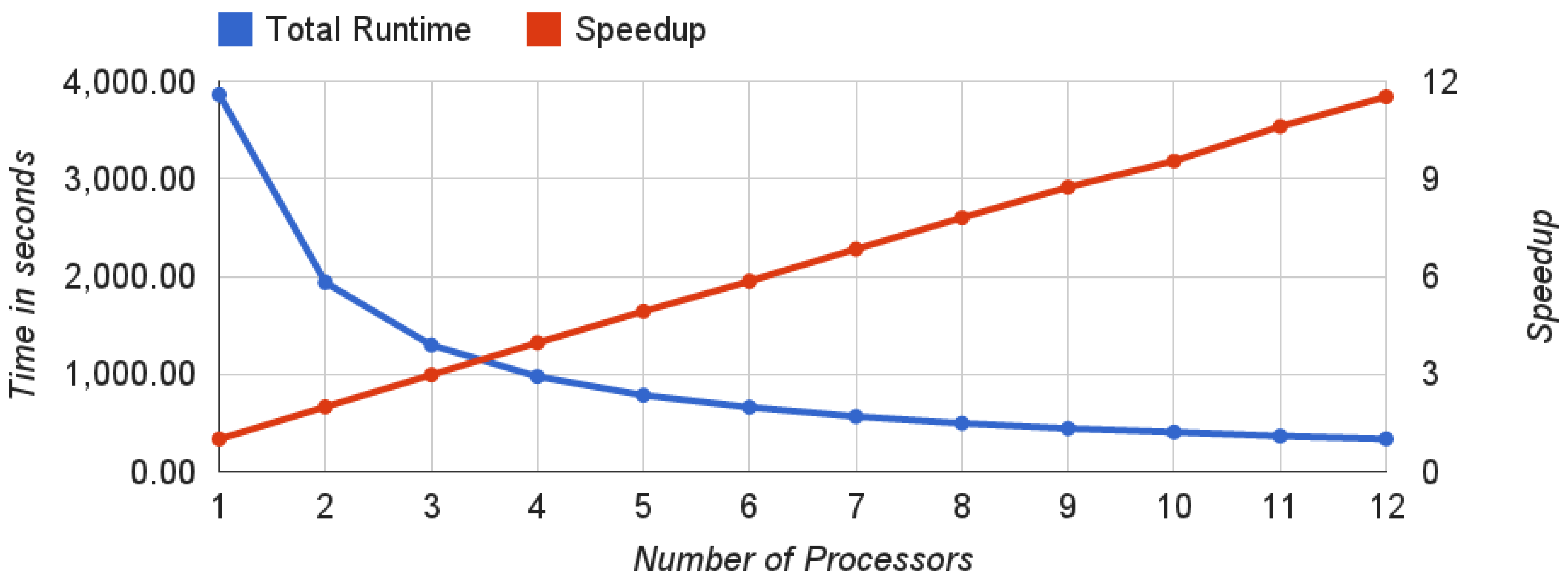
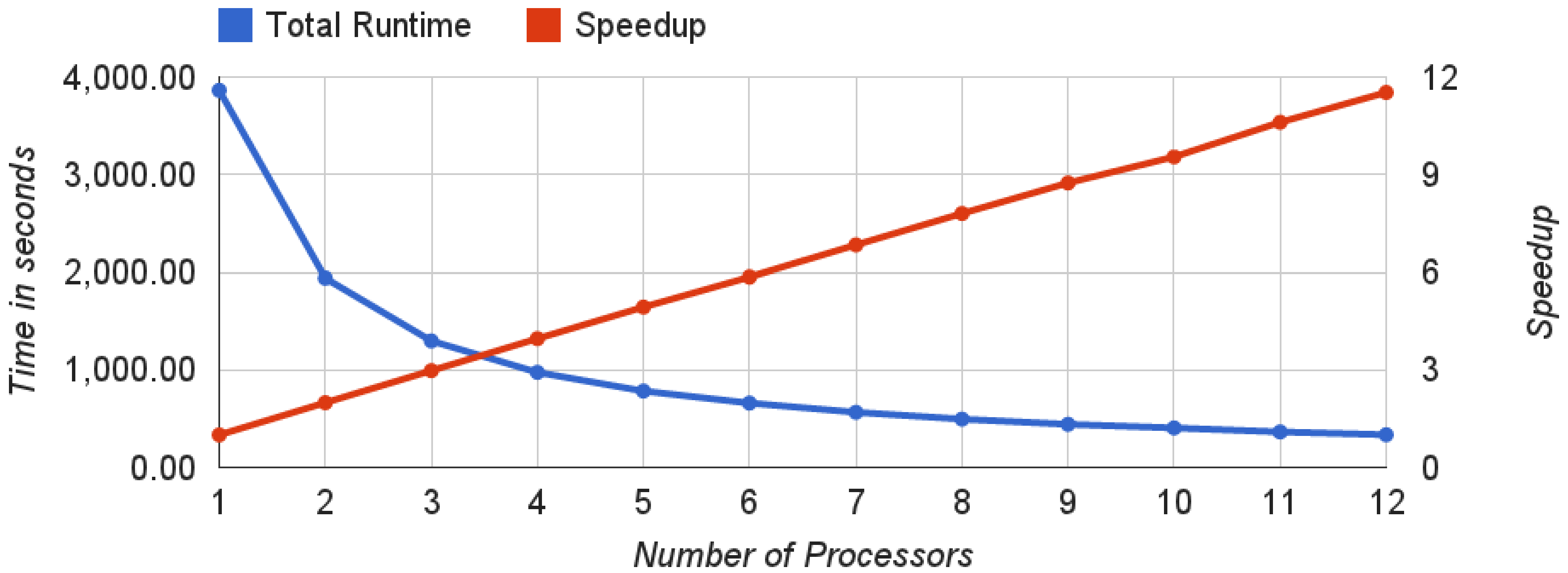


image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$



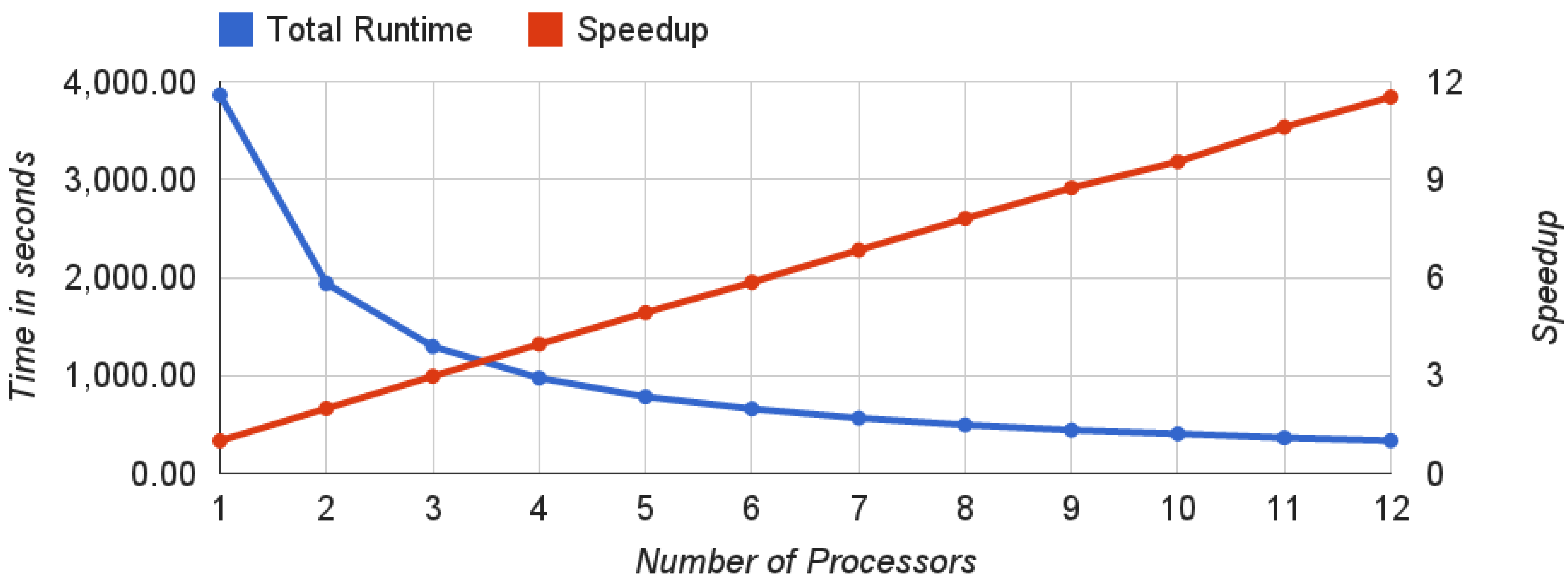
- 335.18 seconds to find collision

image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



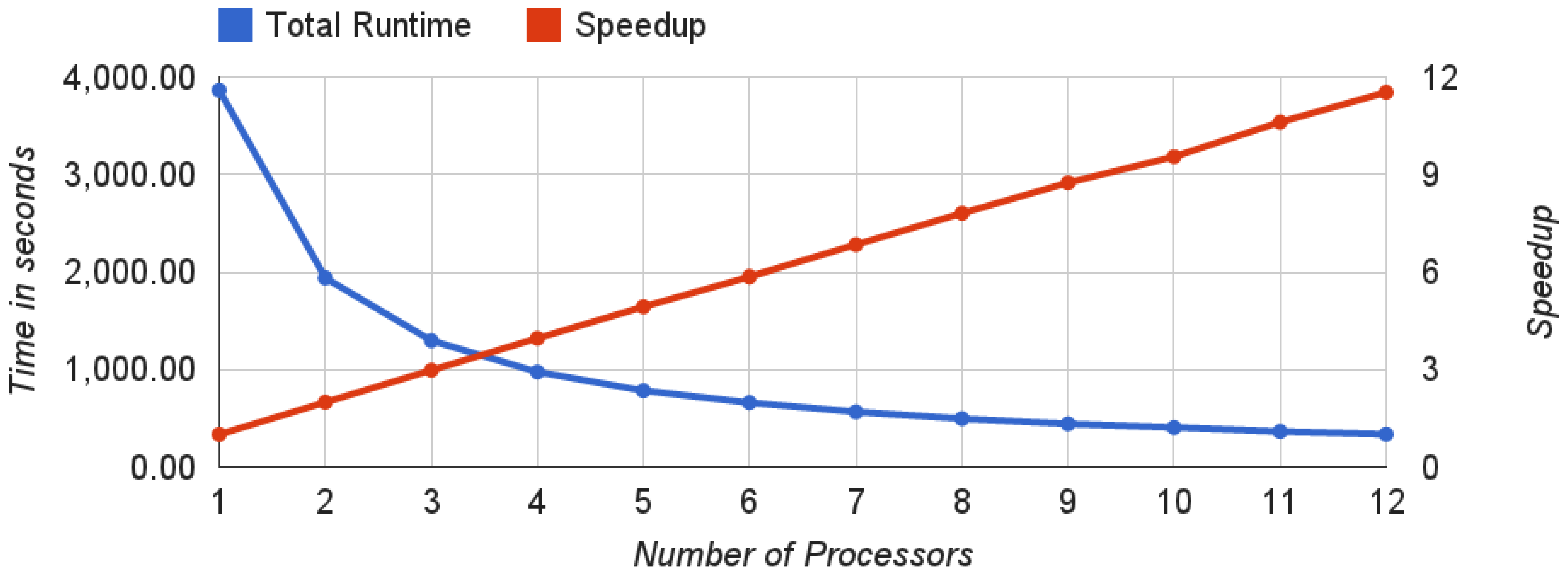
- 335.18 seconds to find collision
- approx. 1 billion strings tried

image source: autoanything

Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



- 335.18 seconds to find collision
- approx. 1 billion strings tried

$\longrightarrow$ linear speedup

image source: autoanything

Benchmarks

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$

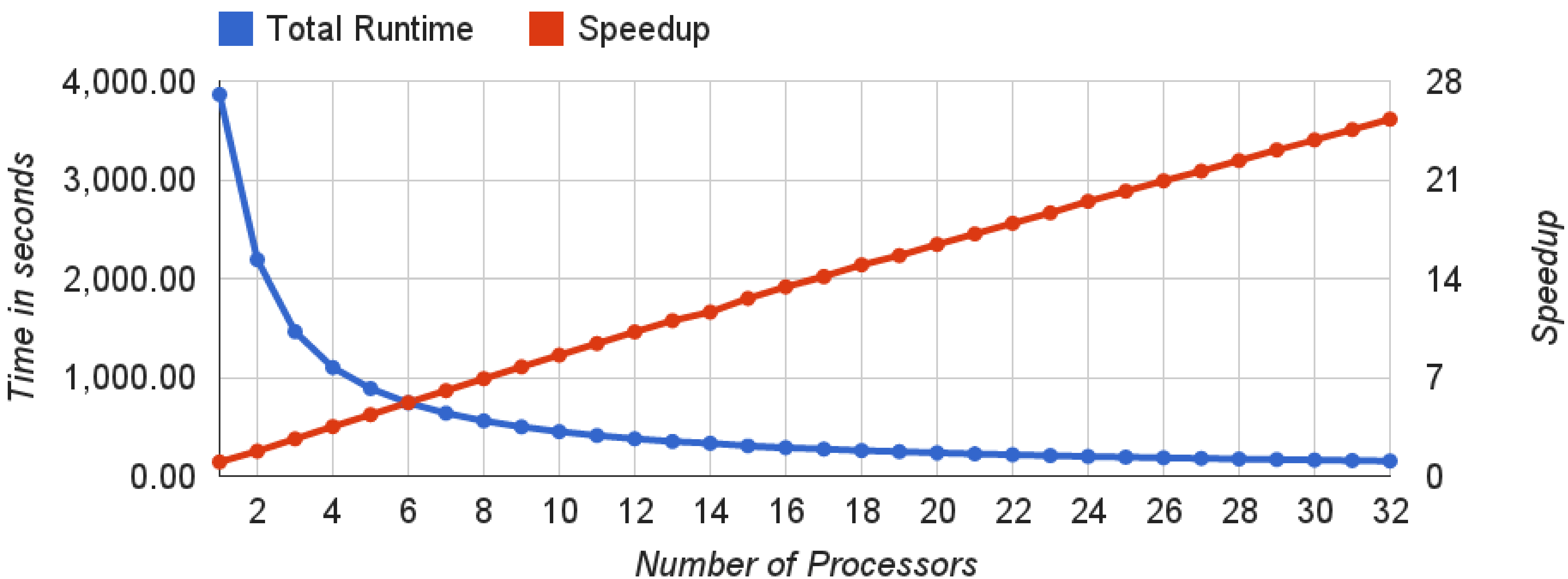
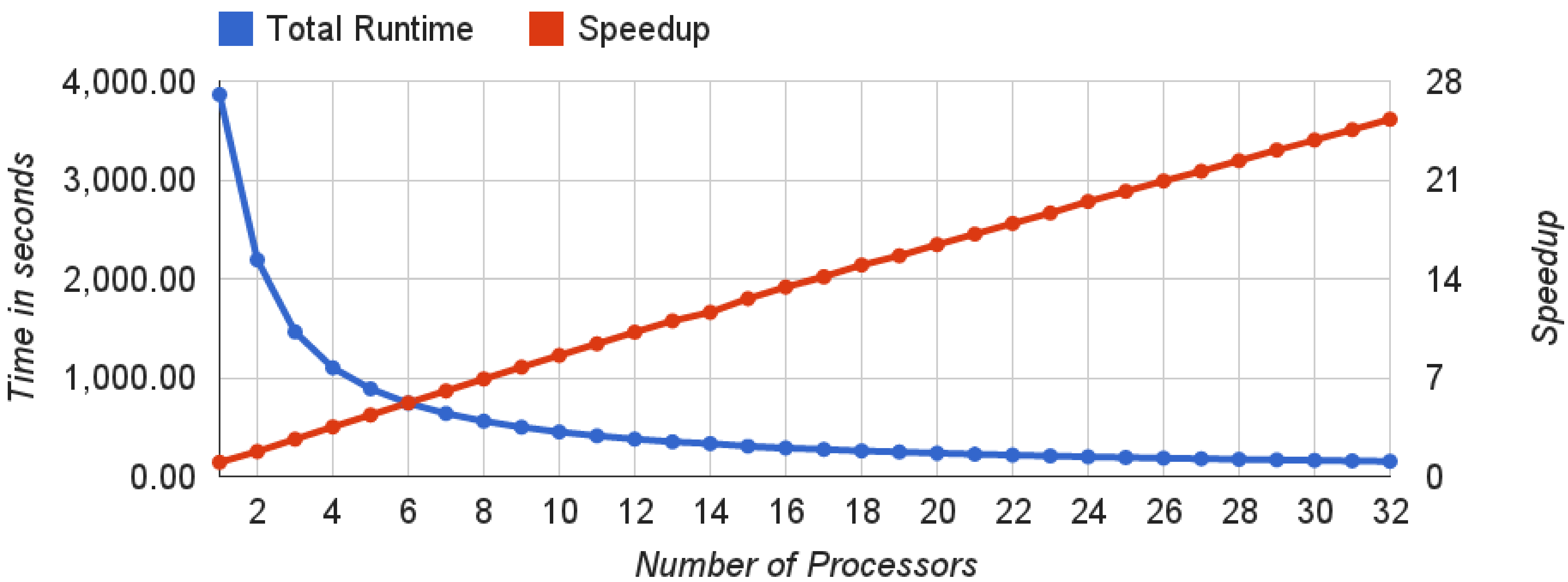


image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



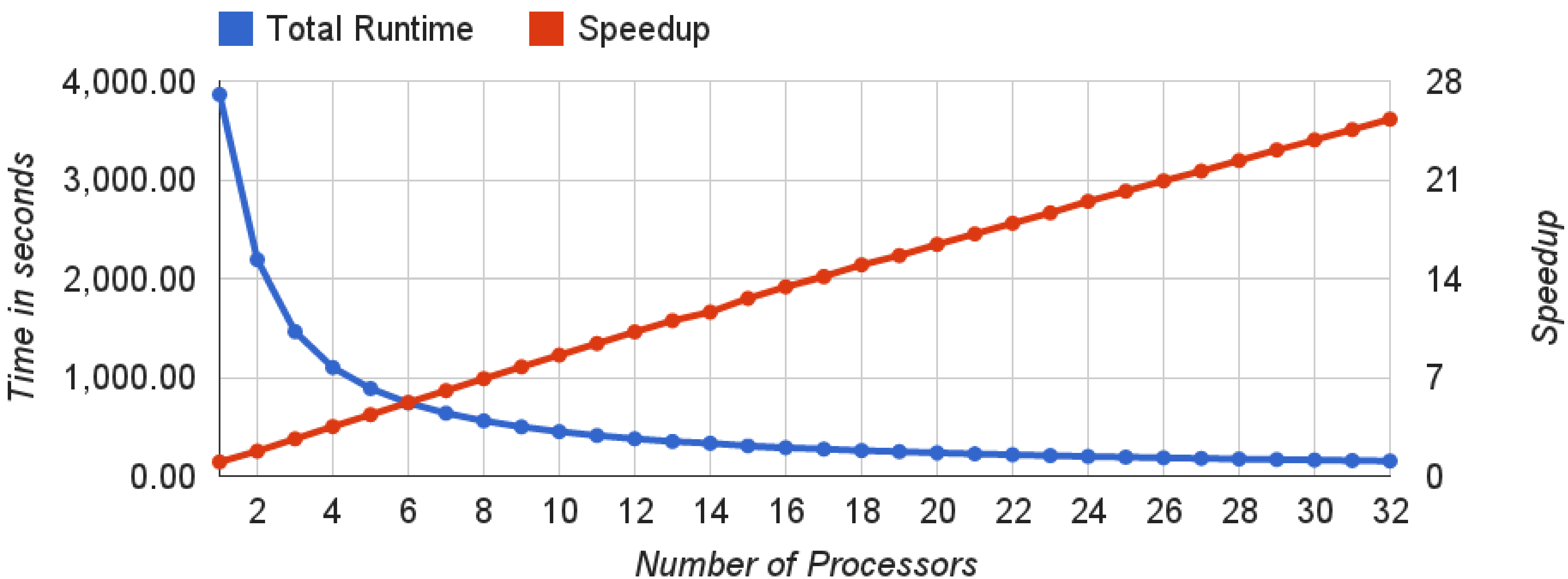
- 152.74 seconds to find collision

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



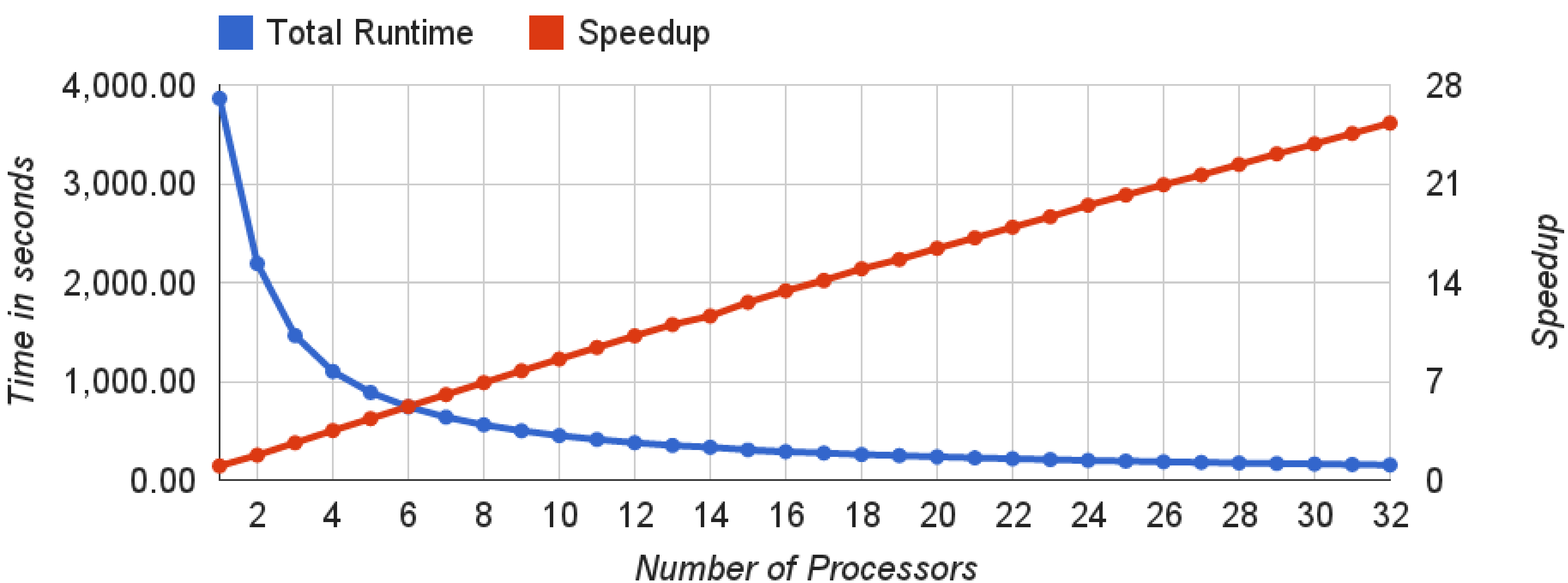
- 152.74 seconds to find collision
- approx. 1 billion strings tried

image source: autoanything

Benchmarks

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



- 152.74 seconds to find collision
- approx. 1 billion strings tried

$\longrightarrow$ linear speedup

image source: autoanything

Benchmarks

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{24}$

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$

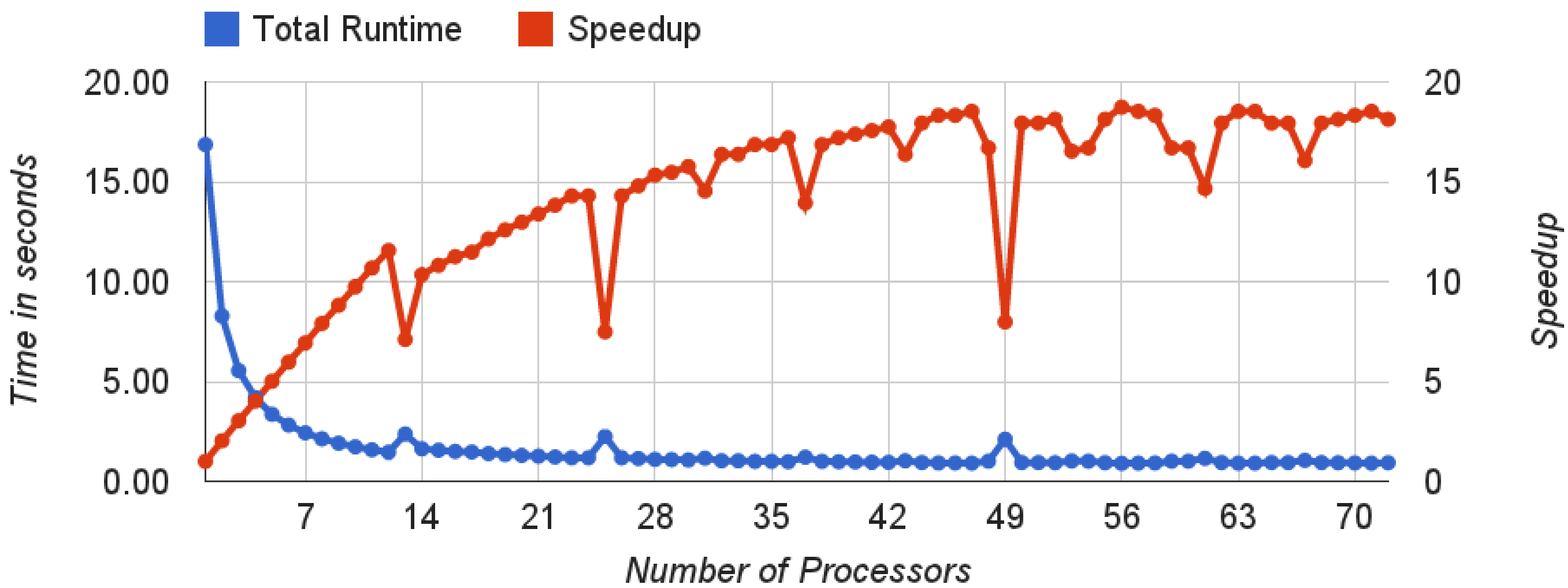
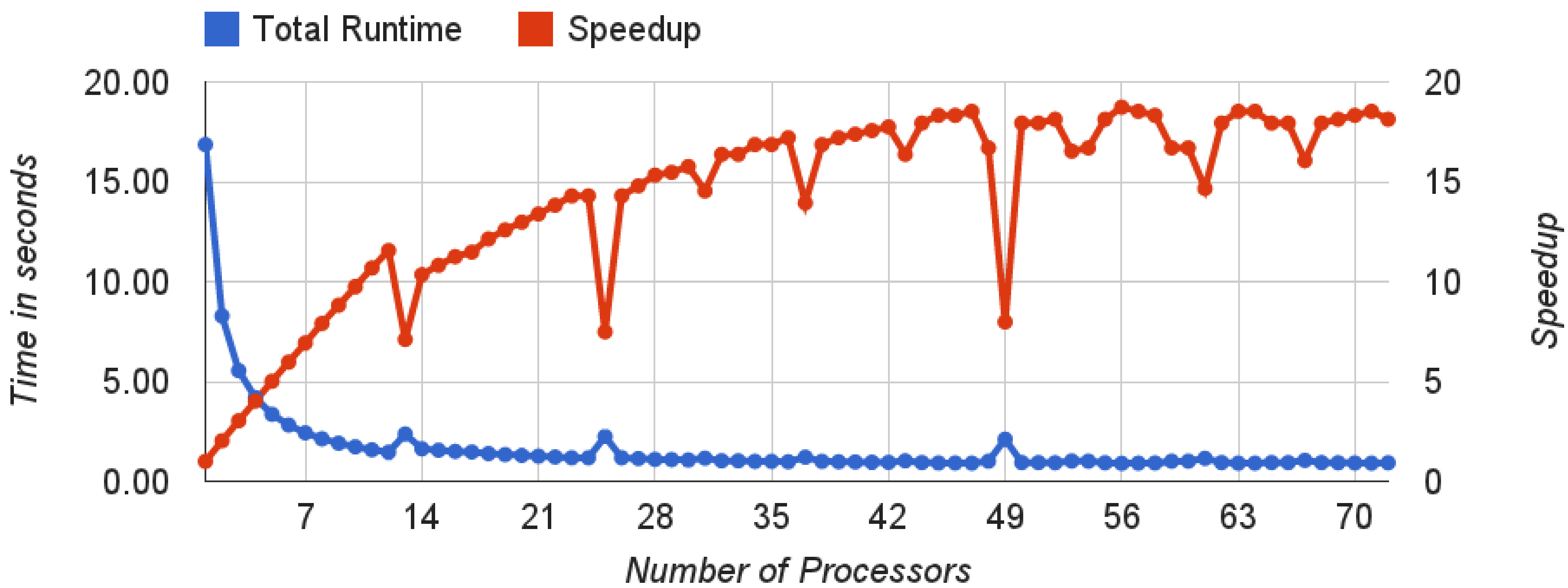


image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{24}$



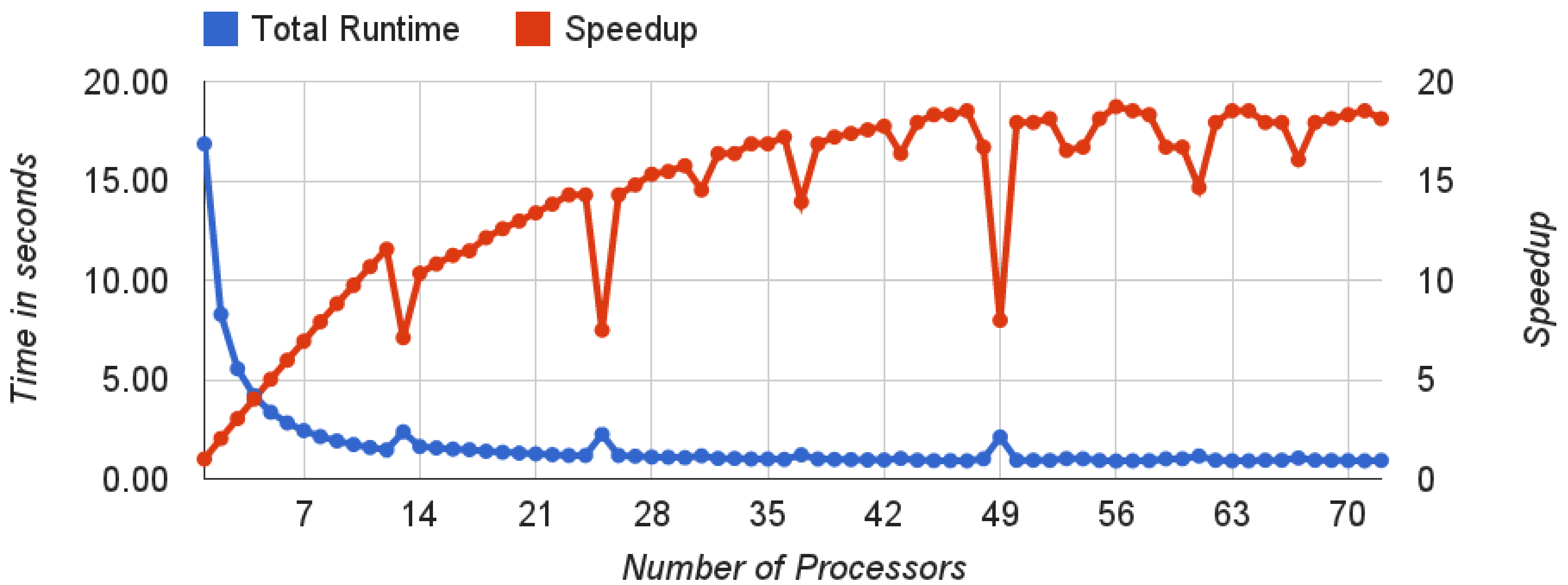
- 0.93 seconds to find collision

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



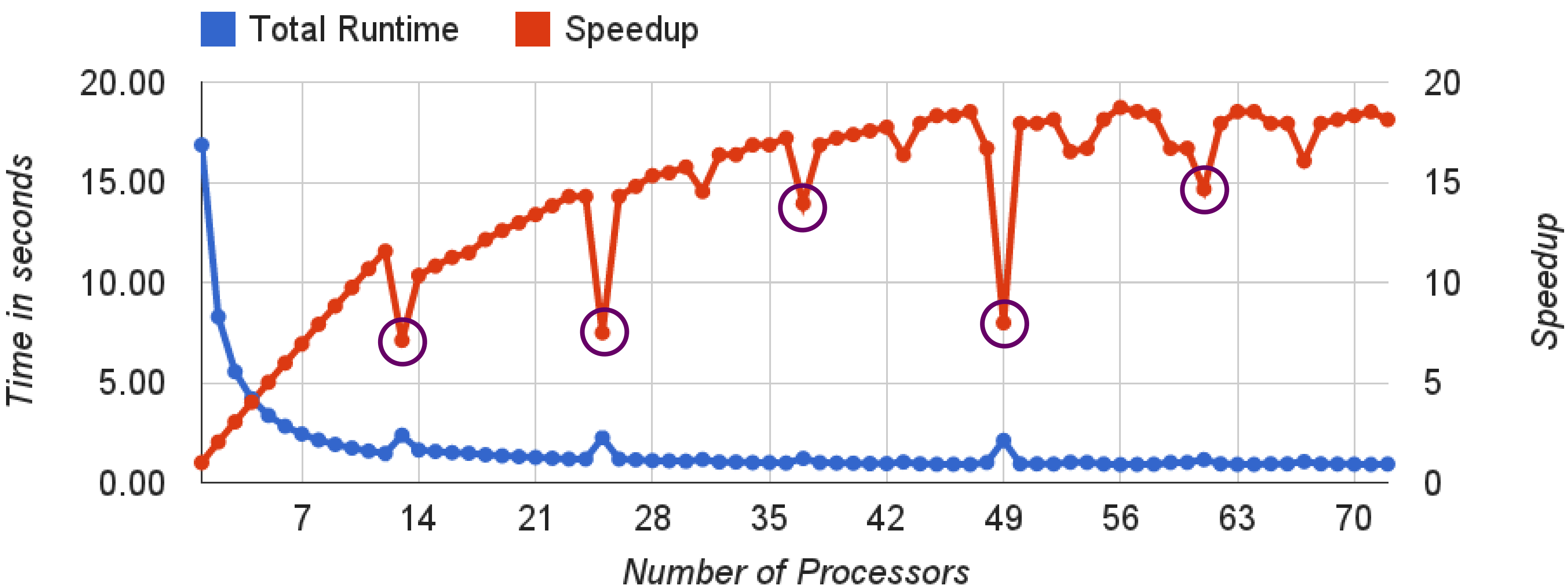
- 0.93 seconds to find collision
- approx. 4 million strings tried

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



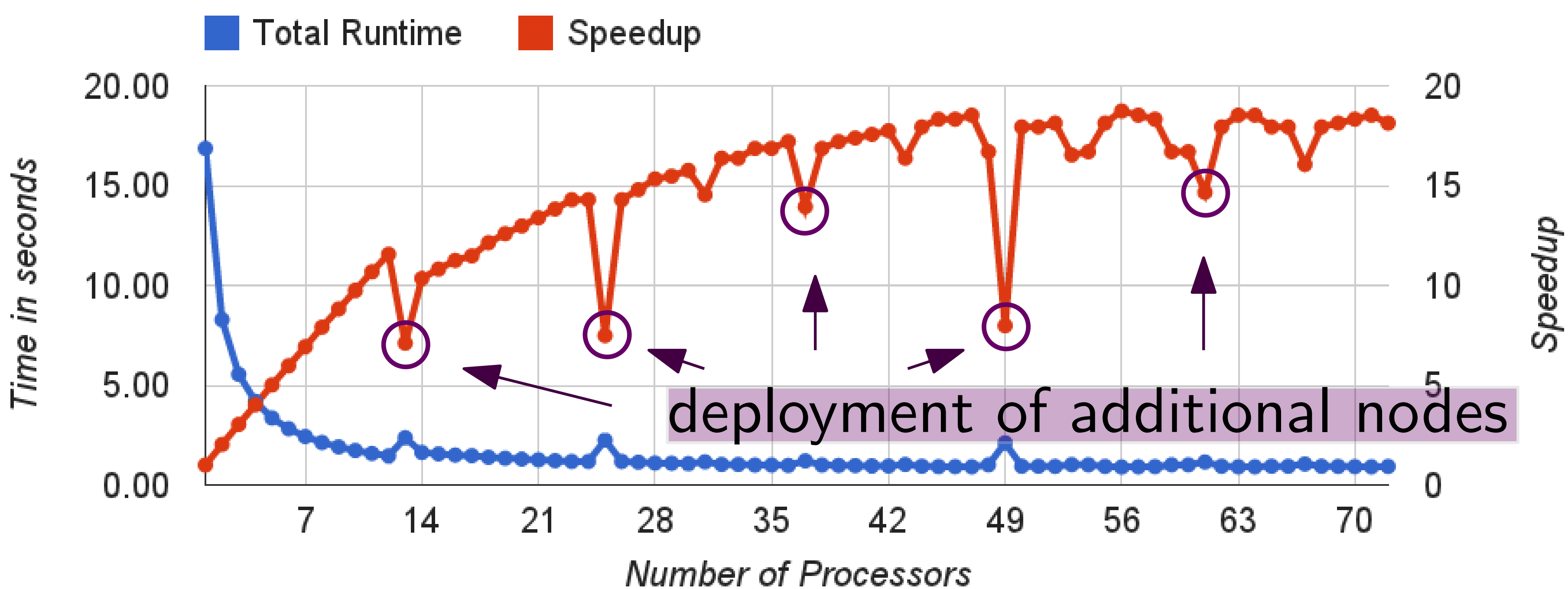
- 0.93 seconds to find collision $\longrightarrow$ speed drop
- approx. 4 million strings tried

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{24}$



- 0.93 seconds to find collision

$\longrightarrow$ speed drop

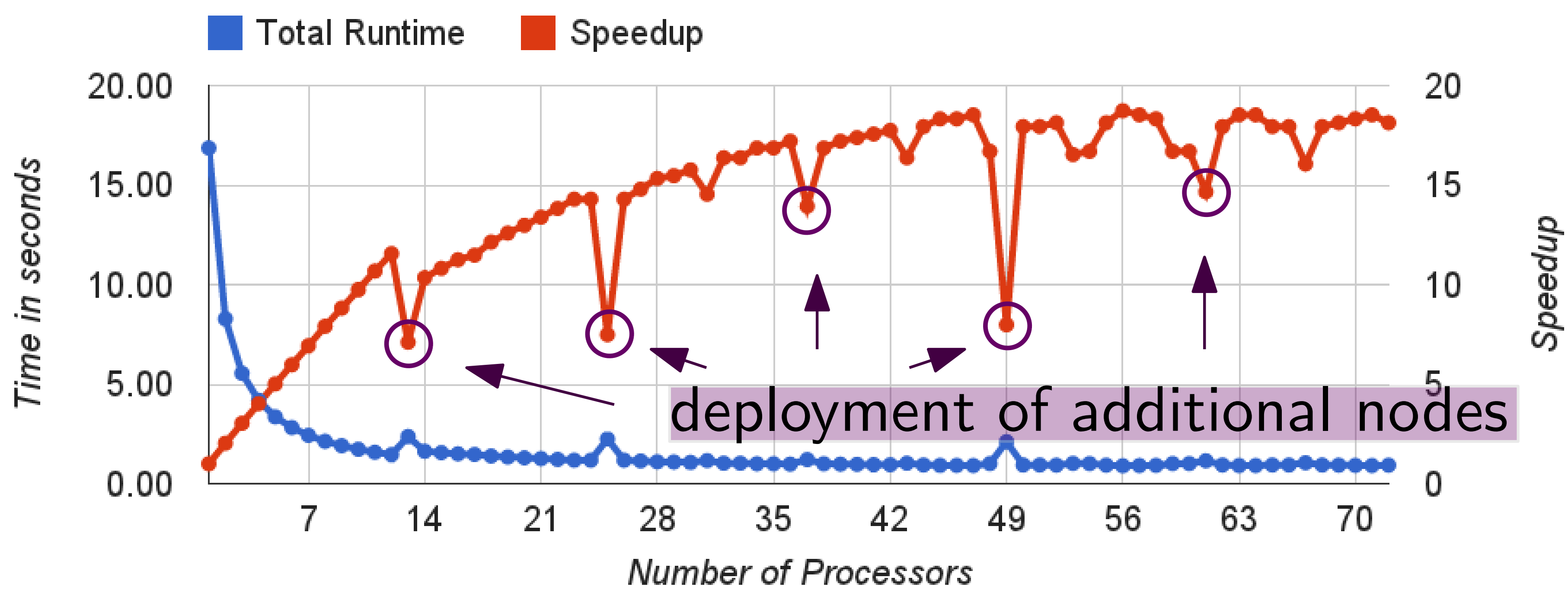
- approx. 4 million strings tried

image source: autoanything

Benchmarks

Fourth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



- 0.93 seconds to find collision
- approx. 4 million strings tried

→ speed drop due to
- small problem size
- setup overhead

image source: autoanything

Benchmarks

image source: autoanything

Benchmarks

Fifth Test: MPI on $6 \cdot 12$ Core Systems

image source: autoanything

Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz

image source: autoanything

Benchmarks

Fifth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$

image source: autoanything

Benchmarks

Fifth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$

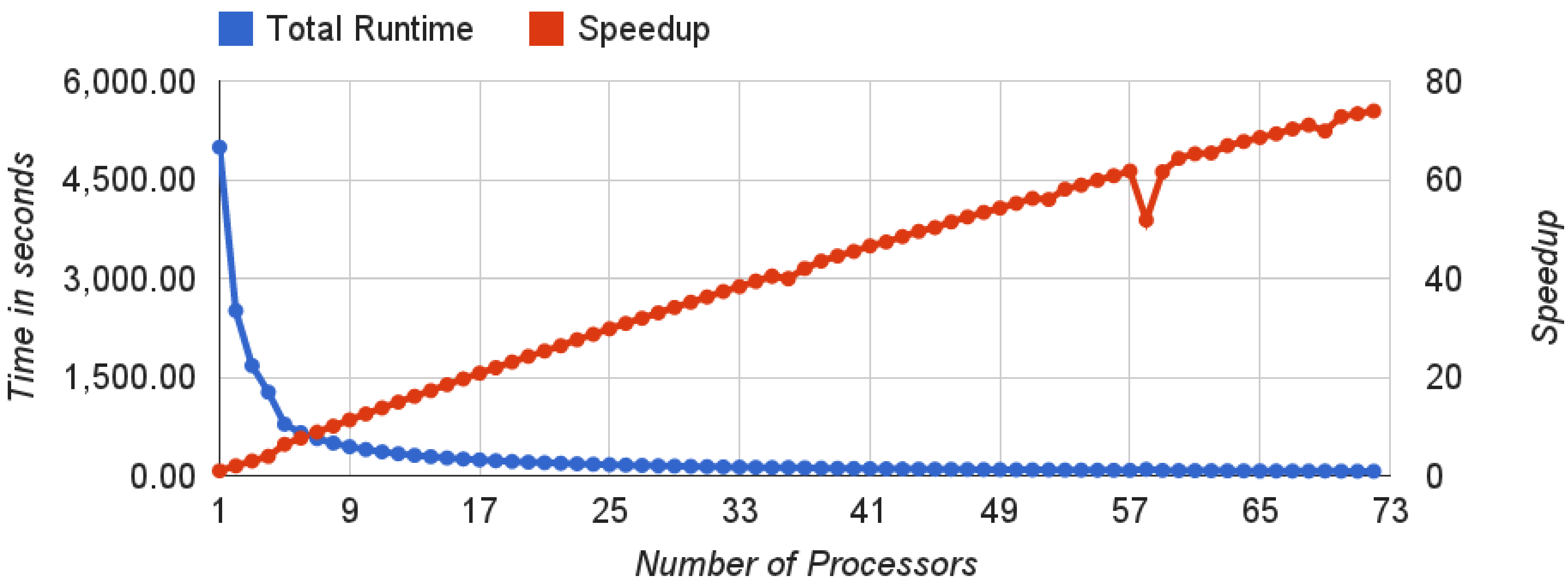
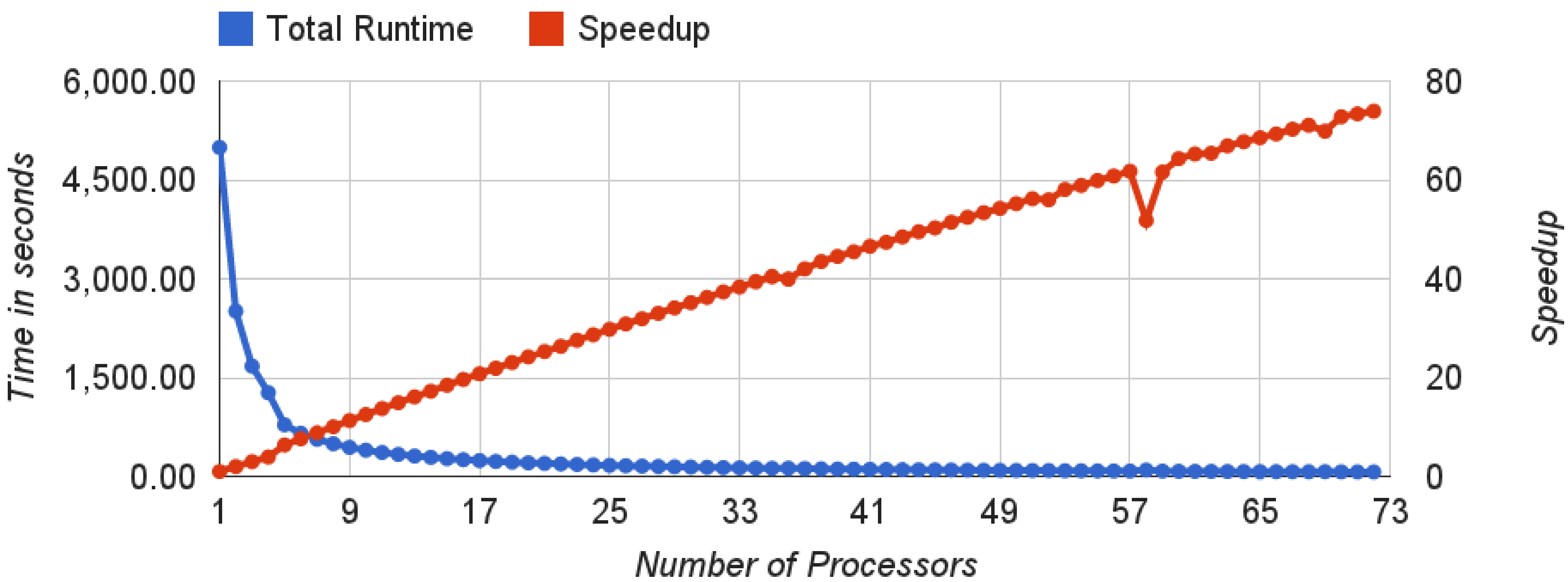


image source: autoanything

Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



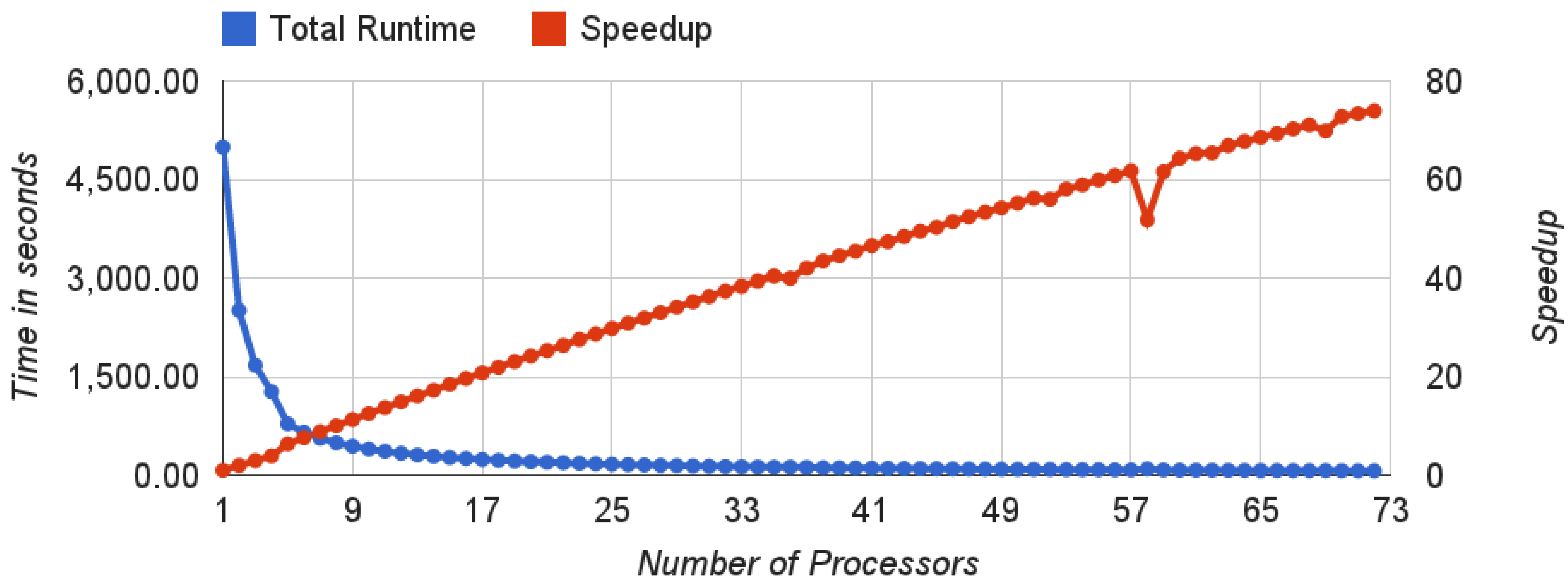
- 67.54 seconds to find collision

image source: autoanything

Benchmarks

Fifth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



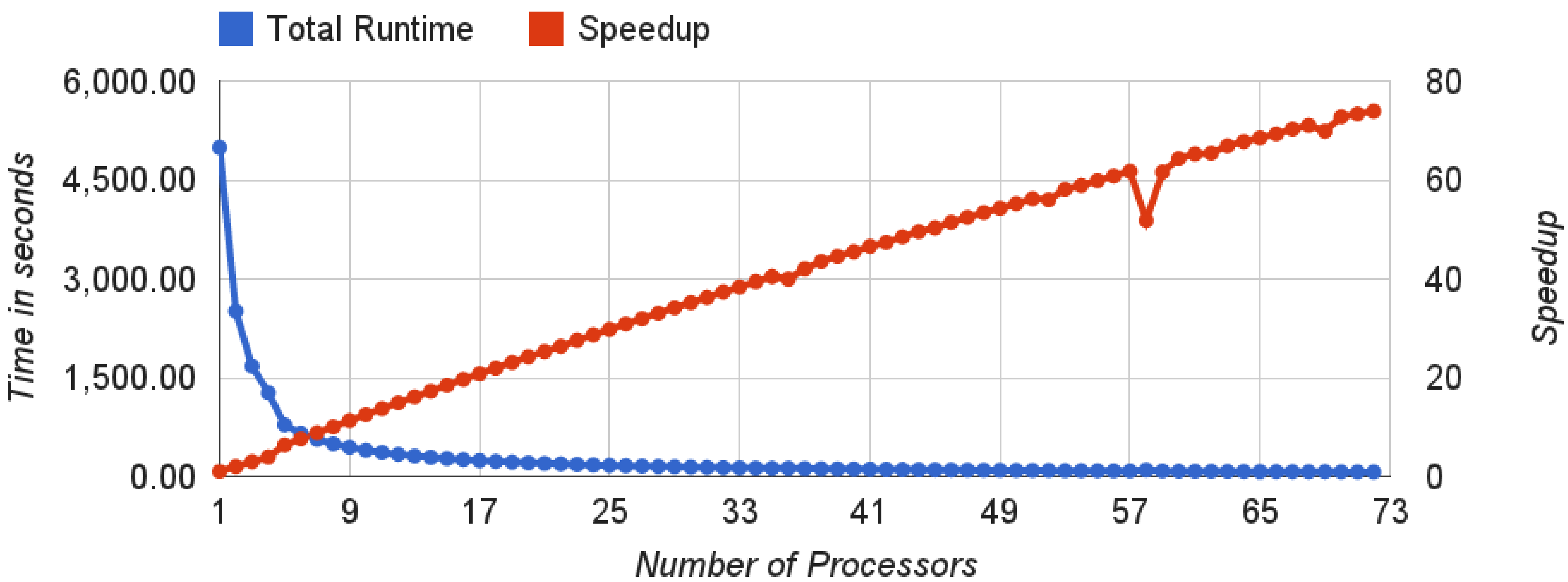
- 67.54 seconds to find collision
- approx. 1 billion strings tried

image source: autoanything

Benchmarks

Fifth Test: MPI on $6 \cdot 12$ Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



- 67.54 seconds to find collision
- approx. 1 billion strings tried

$\longrightarrow$ linear speedup

image source: autoanything

Benchmarks

image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

image source: autoanything

Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems

image source: autoanything

Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz

image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0, 1\}^{24}$

image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{24}$

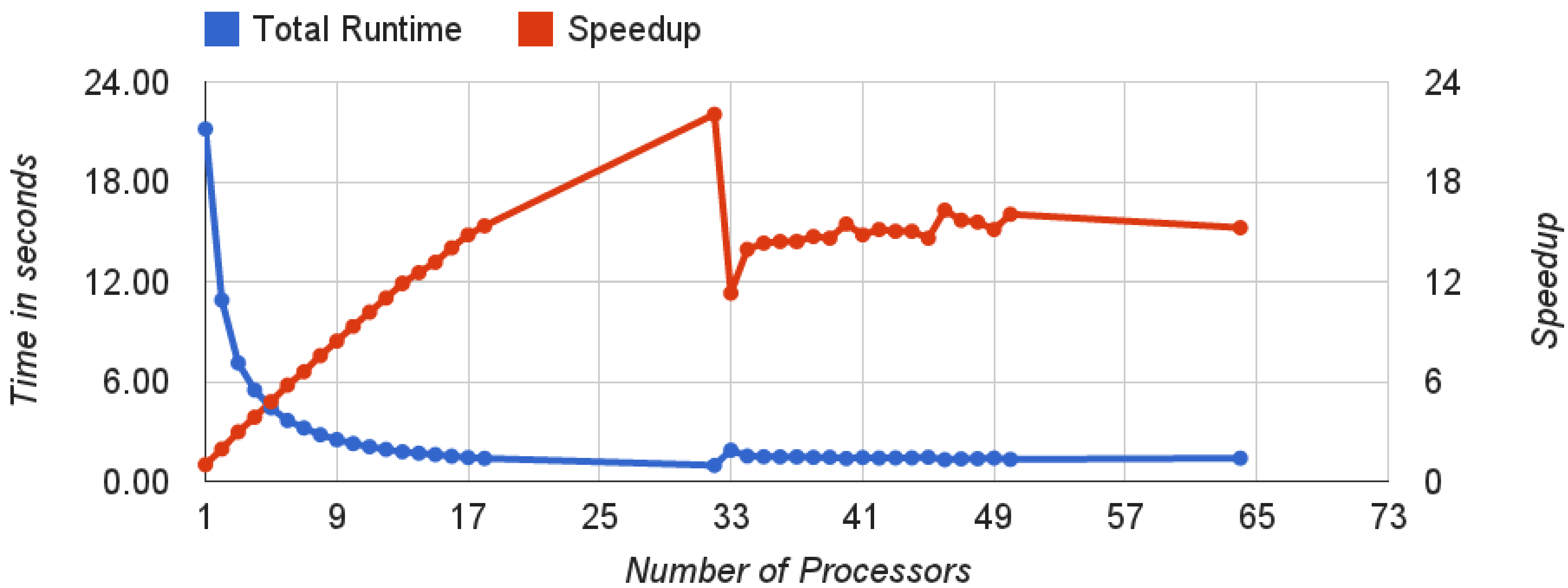
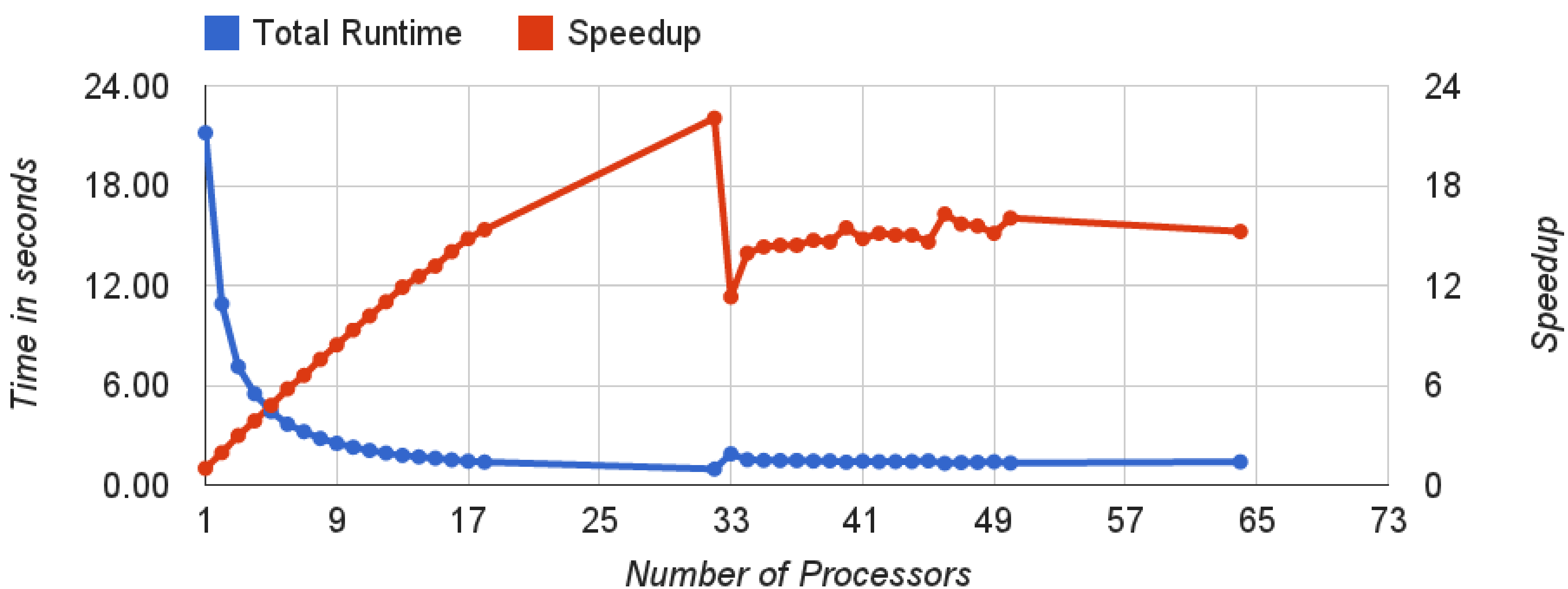


image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



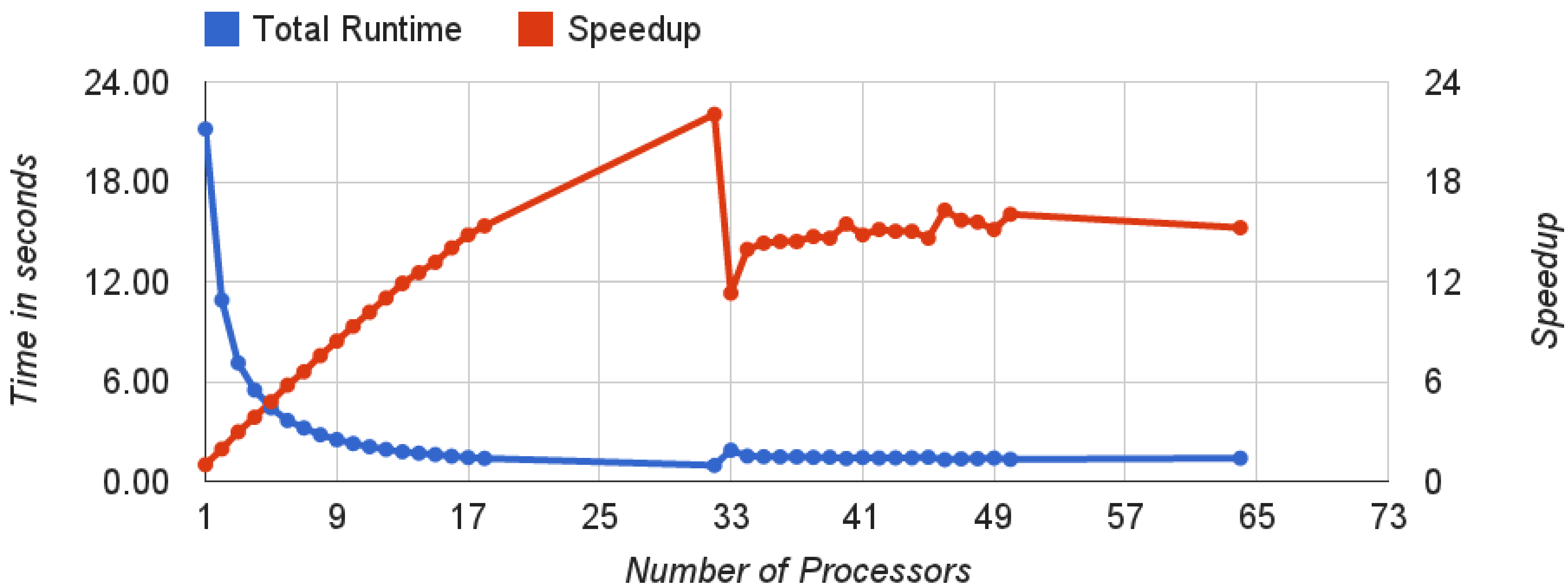
- 0.96 seconds to find collision

image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0, 1\}^{24}$



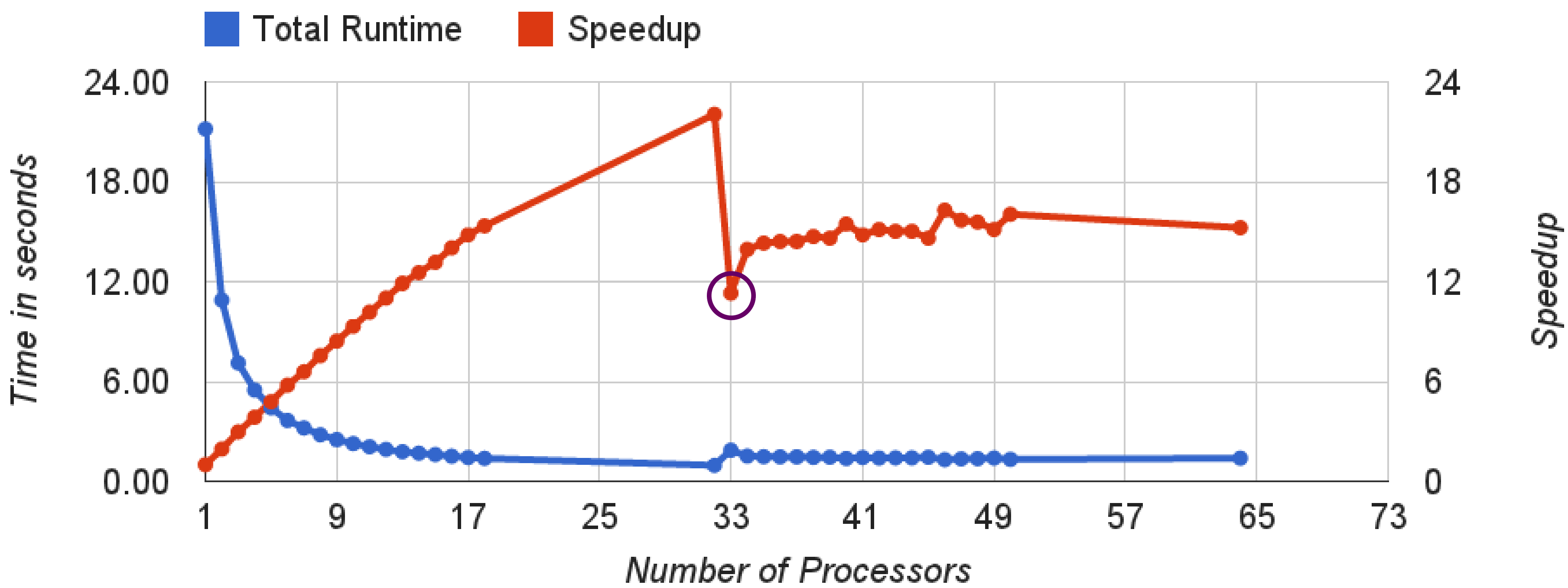
- 0.96 seconds to find collision
- approx. 4 million strings tried

image source: autoanything

Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



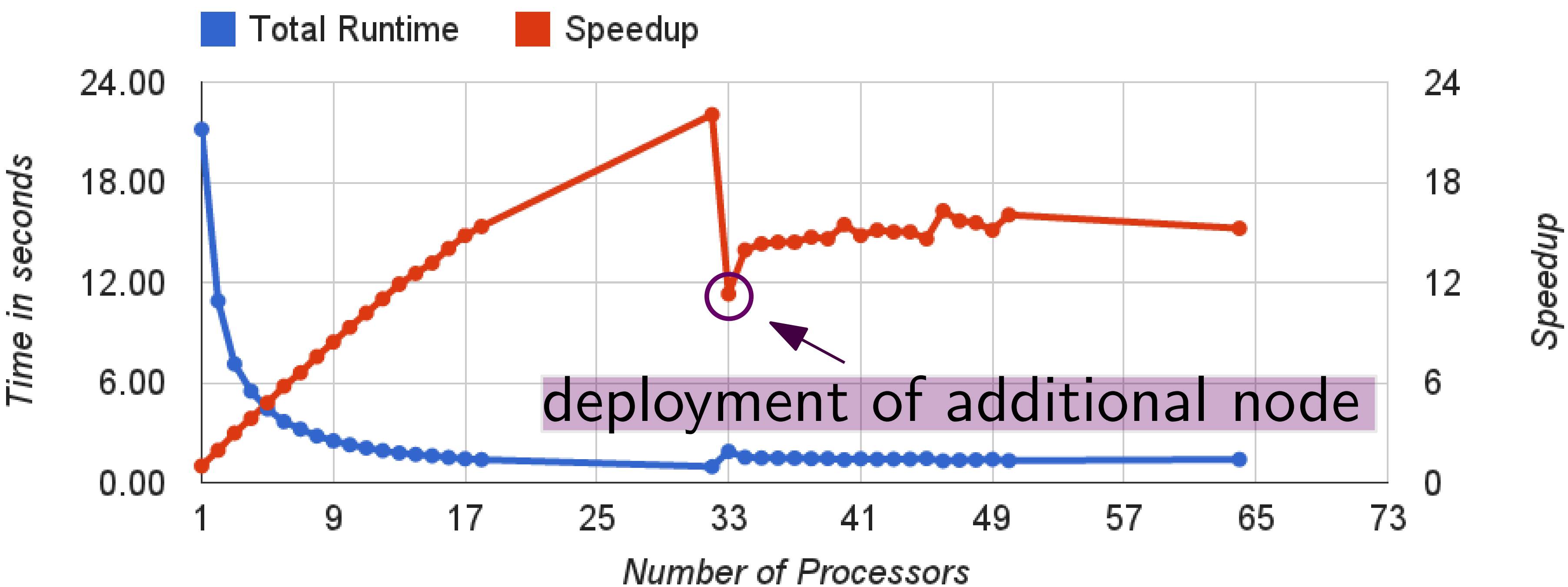
- 0.96 seconds to find collision → speed drop
- approx. 4 million strings tried

image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



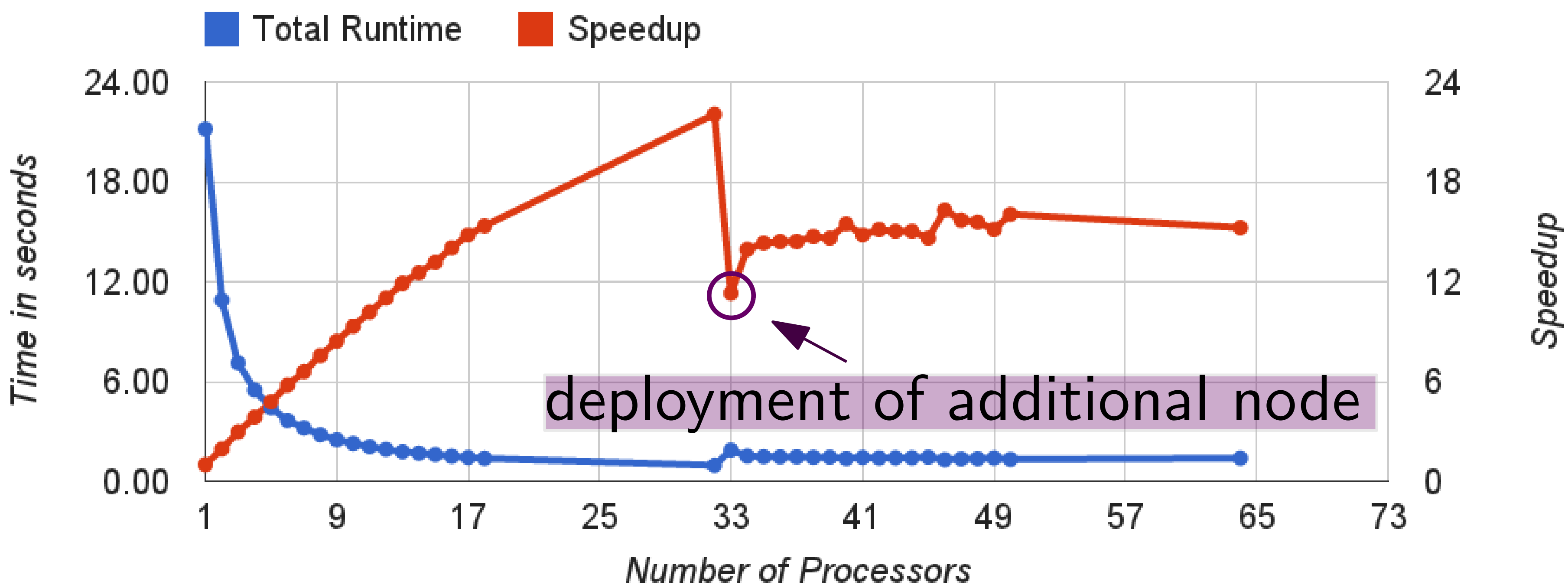
- 0.96 seconds to find collision → speed drop due to
- approx. 4 million strings tried

image source: autoanything

Benchmarks

Sixth Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{24}$



- 0.96 seconds to find collision
- approx. 4 million strings tried

→ speed drop due to

- small problem size
- setup overhead

image source: autoanything

Benchmarks

image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz

image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$

image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$

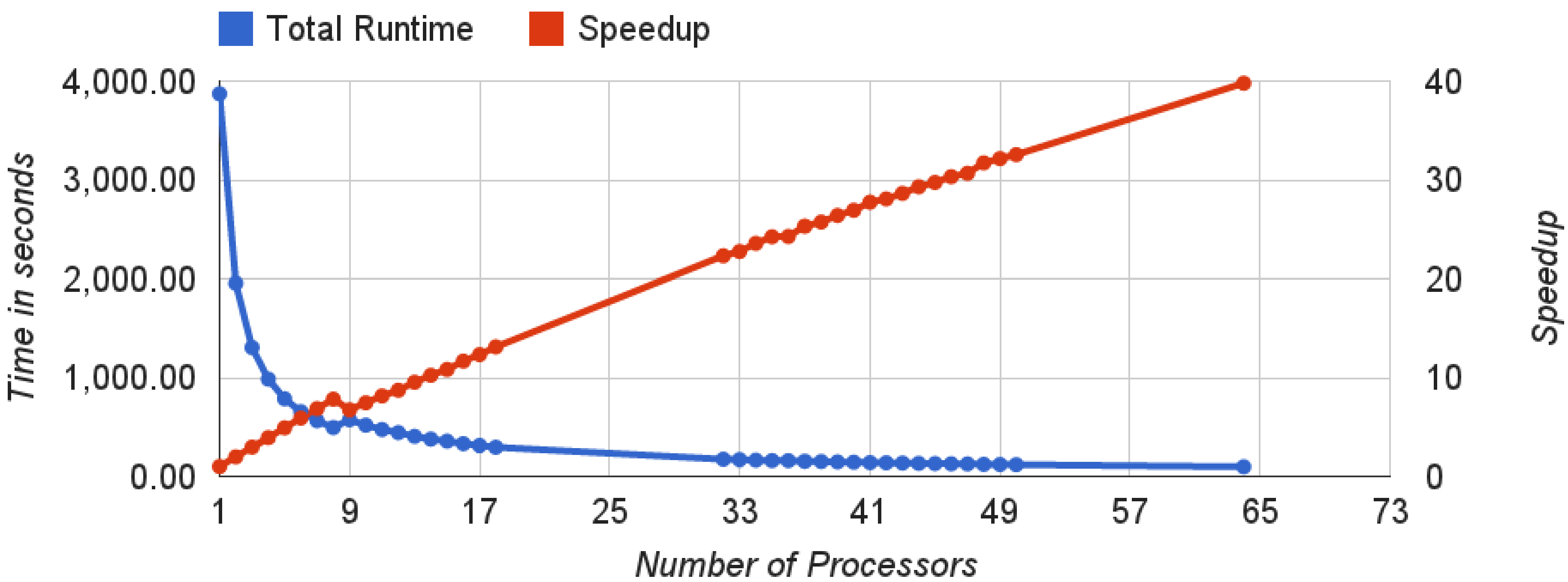
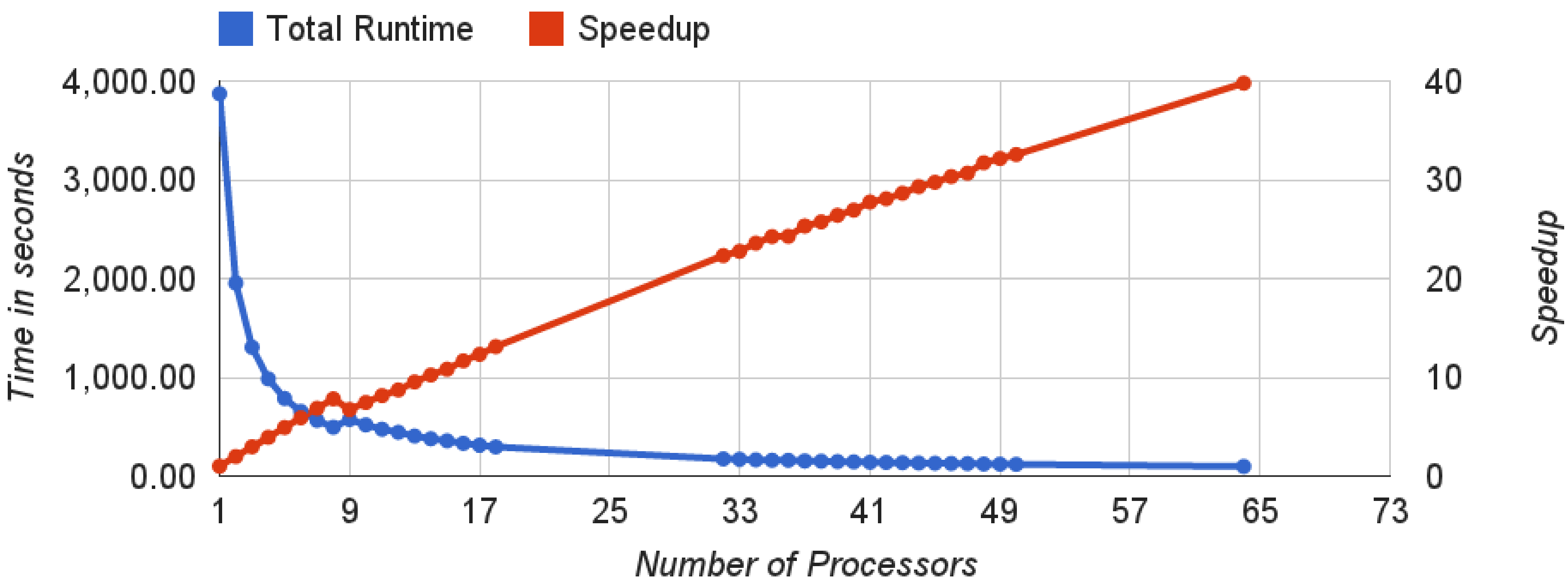


image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



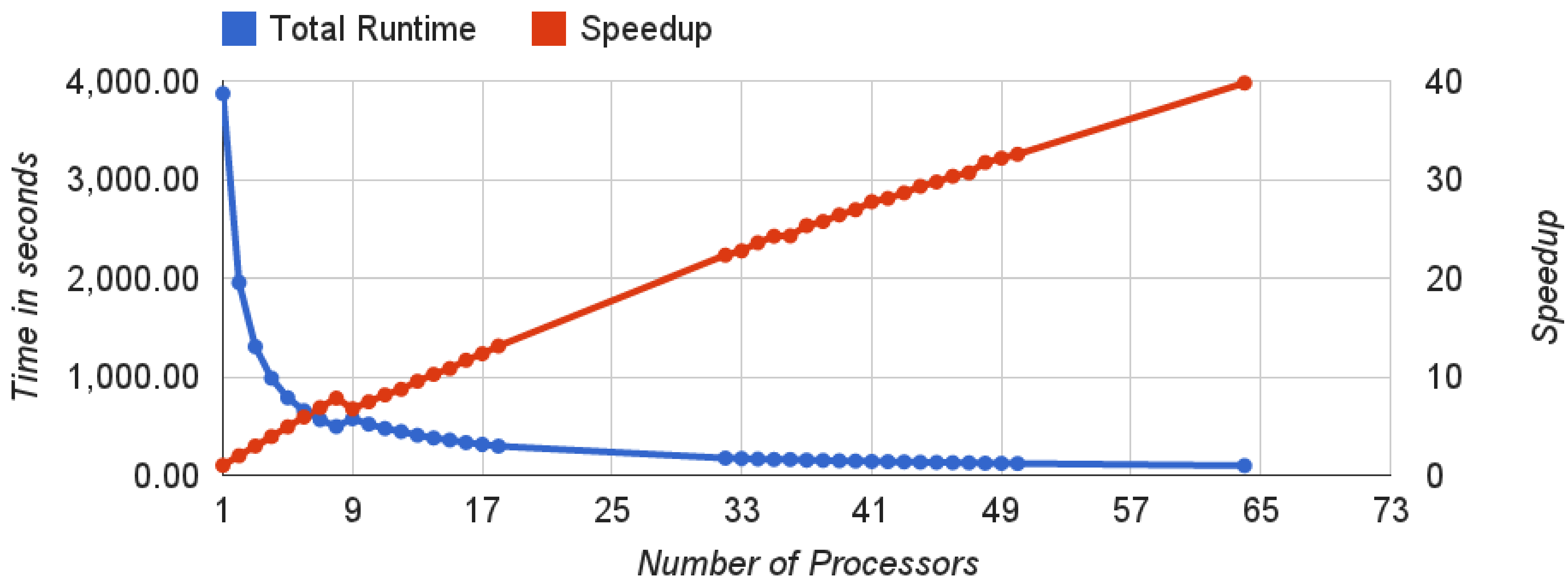
- 97.30 seconds to find collision

image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0,1\}^{32}$



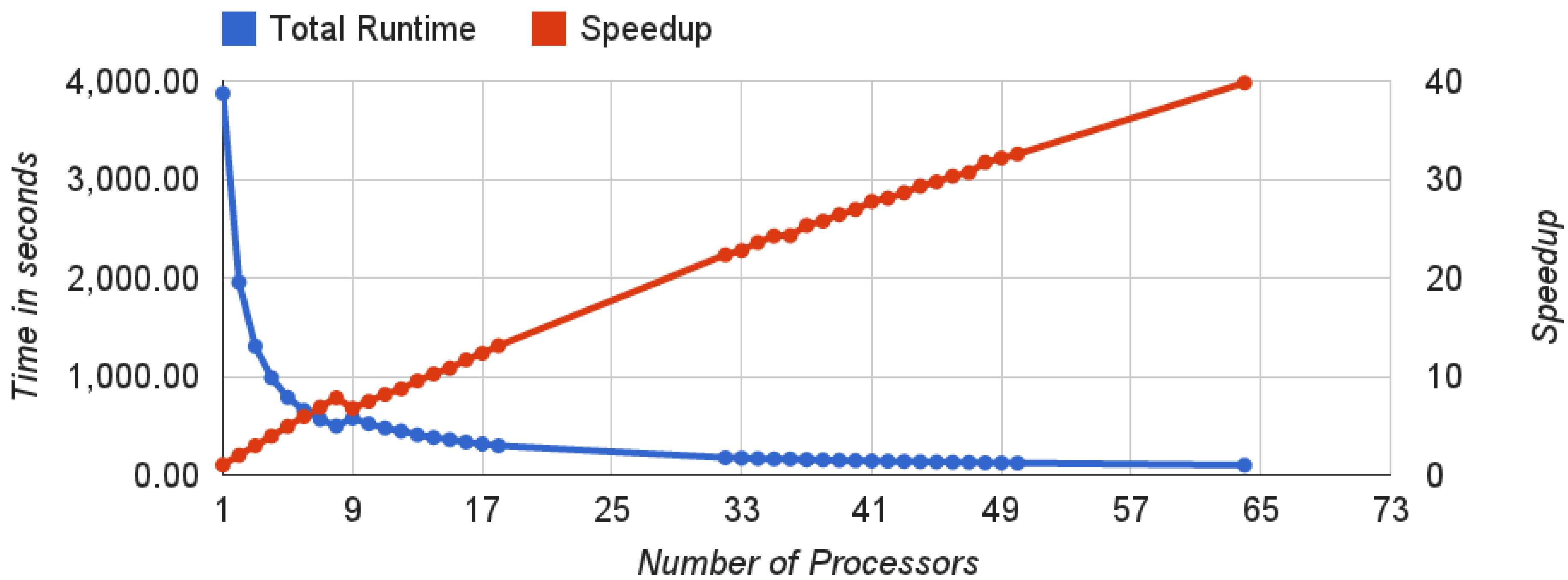
- 97.30 seconds to find collision
- approx. 1 billion strings tried

image source: autoanything

Benchmarks

Seventh Test: MPI on $2 \cdot 32$ Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: md5(x) with $x \in \{0, 1\}^{32}$



- 97.30 seconds to find collision
- approx. 1 billion strings tried

→ linear speedup

image source: autoanything

Explanations and Expectations

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

    but: already used CCR's "biggest" machine

MPI implementation

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare $x \in \{0,1\}^{32}$

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare $x \in \{0,1\}^{32}$

OpenMP

MPI

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare $x \in \{0,1\}^{32}$

# PEs	32
OpenMP	
MPI	

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare $x \in \{0,1\}^{32}$

# PEs	32
OpenMP	152s
MPI	174s

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare $x \in \{0,1\}^{32}$

# PEs	32	64
OpenMP	152s	
MPI	174s	

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

compare $x \in \{0,1\}^{32}$

# PEs	32	64
OpenMP	152s	–
MPI	174s	97s

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

CUDA implementation

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

CUDA implementation (future work)

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

  but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

  suitable for larger problem sizes

CUDA implementation (future work)

- fast communication between processing elements

image source: ozbinoculars

Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
- suitable for smaller problem sizes

but: already used CCR's "biggest" machine

MPI implementation

- slower communication, setup overhead
- more processors available

suitable for larger problem sizes

CUDA implementation (future work)

- fast communication between processing elements
- very high number of processors on single nodes

image source: ozbinoculars

Sources and References

Hans Delf and Helmut Knebl. *Introduction to Cryptography: Principles and Applications.* Springer, 2007.

Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

Charles P. Pfleeger and Shari L. Pfleeger. *Analyzing Computer Security: A Threat/Vulnerability/Countermeasure Approach* Prentice Hall, 2011.