CSE 633: Parallel Algorithms
Fall 2012

Parallelized Hash Collision Attacking

Benedikt Budig

Course Instructor:
Russ Miller
The Topic in a Nutshell
The Topic in a Nutshell

A hash function is a total function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$
The Topic in a Nutshell

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

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

Project Goal: Find Hash Collisions for given Hash
The Topic in a Nutshell

A hash function is a total function \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \) 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 \( h(x) = h(y) \),
The Topic in a Nutshell

A *hash function* is a total function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ 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.
The Topic in a Nutshell

A hash function is a total function \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \) 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.

\[ \text{find a string } y \text{ such that } h(y) \text{ collides with given } h(x) \]
The Topic in a Nutshell

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

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

Reason to do that?
The Topic in a Nutshell

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

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

Reason to do that: Cryptographic Applications
The Topic in a Nutshell

A hash function is a total function \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \) 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.

\[ \text{find a string } y \text{ such that } h(y) \text{ collides with given } h(x) \]

Reason to do that: Cryptographic Applications

- secure storage of passwords
The Topic in a Nutshell

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

→ 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
The Topic in a Nutshell

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

→ 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 → MD5
Parallel Approach
Parallel Approach

Input:
Parallel Approach

Input: hash $\text{md5}(x)$ of unknown string $x$ of length $|x| \leq n$
Parallel Approach

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

Output:
Parallel Approach

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

Output: string $y$ such that $\text{md5}(y) = \text{md5}(x)$ and $|y| \leq n$
Parallel Approach

Input: \( \text{hash } \text{md5}(x) \) of unknown string \( x \) of length \( |x| \leq n \)

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

Parallel approach
Parallel Approach

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

**Output:** string $y$ such that $\text{md5}(y) = \text{md5}(x)$ 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 $\text{md5}(x)$ of unknown string $x$ of length $|x| \leq n$

Output: string $y$ such that $\text{md5}(y) = \text{md5}(x)$ 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 $\text{md5}(x)$ of unknown string $x$ of length $|x| \leq n$

Output: string $y$ such that $\text{md5}(y) = \text{md5}(x)$ 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 \( \text{md5}(x) \) of unknown string \( x \) of length \(|x| \leq n\)

Output: string \( y \) such that \( \text{md5}(y) = \text{md5}(x) \) 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 $\text{md5}(x)$ of unknown string $x$ of length $|x| \leq n$

Output: string $y$ such that $\text{md5}(y) = \text{md5}(x)$ 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 $\text{md5}(x)$ of unknown string $x$ of length $|x| \leq n$

Output: string $y$ such that $\text{md5}(y) = \text{md5}(x)$ 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 \( \text{md5}(x) \) of unknown string \( x \) of length \( |x| \leq n \)

Output: string \( y \) such that \( \text{md5}(y) = \text{md5}(x) \) 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
Technical Realization

Hardware

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

image source: CCR website
Technical Realization

Hardware

- use of CPUs
- use of a multi-core system
- InfiniBand network
Technical Realization

Hardware

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

**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

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

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

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

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

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

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
Benchmarks

First Test: OpenMP on 12 Core System
Benchmarks

First Test: OpenMP on 12 Core System

• 12 Intel Xeon E5645 at 2.40GHz
Benchmarks

First Test: OpenMP on 12 Core System

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

First Test: OpenMP on 12 Core System

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

First Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
- input: md5(x) with x ∈ \{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: $\text{md5}(x)$ with $x \in \{0, 1\}^{24}$
- 1.43 seconds to find collision
- approx. 4 million strings tried
Benchmarks

First Test: OpenMP on 12 Core System

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

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

Image source: autoanything
Benchmarks

image source: autoanything
Benchmarks

Second Test: OpenMP on 12 Core System
Benchmarks

Second Test: OpenMP on 12 Core System

- 12 Intel Xeon E5645 at 2.40GHz
Benchmarks

Second Test: OpenMP on 12 Core System

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

Second Test: OpenMP on 12 Core System

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

Second Test: OpenMP on 12 Core System

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

- 335.18 seconds to find collision
Benchmarks

Second Test: OpenMP on 12 Core System

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

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

linear speedup

image source: autoanything
Benchmarks
Benchmarks

Third Test: OpenMP on 32 Core System
Benchmarks

Third Test: OpenMP on 32 Core System
Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
Benchmarks

Third Test: OpenMP on 32 Core System

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

Third Test: OpenMP on 32 Core System

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

Third Test: OpenMP on 32 Core System

- 32 Intel Xeon E7-4830 at 2.13GHz
- input: $\text{md5}(x)$ with $x \in \{0, 1\}^{32}$
- 152.74 seconds to find collision
- approx. 1 billion strings tried
Benchmarks

Third Test: OpenMP on 32 Core System

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

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

Image source: autoanything
Benchmarks
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems

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

Fourth Test: MPI on 6 \cdot 12 Core Systems

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

Fourth Test: MPI on 6 · 12 Core Systems

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

• 0.93 seconds to find collision
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: $\text{md5}(x)$ with $x \in \{0, 1\}^{24}$
- 0.93 seconds to find collision
- approx. 4 million strings tried
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: $\text{md5}(x)$ 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 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: $\text{md5}(x)$ with $x \in \{0, 1\}^{24}$
- 0.93 seconds to find collision
- approx. 4 million strings tried

![Graph showing total runtime and speedup vs. number of processors.]

- Deployment of additional nodes
- Speed drop

*Image source: autoanything*
Benchmarks

Fourth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
- input: \( \text{md5}(x) \) with \( x \in \{0, 1\}^{24} \)
  
  - 0.93 seconds to find collision
  - approx. 4 million strings tried

![Graph showing time in seconds vs. number of processors, with Speedup and Total Runtime lines. The graph highlights deployment of additional nodes and speed drop due to small problem size and setup overhead.]
Benchmarks

image source: autoanything
Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems
Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems

- 6 hosts with 12 Intel Xeon E5645 at 2.40GHz
Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems

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

Fifth Test: MPI on 6 · 12 Core Systems

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

![Graph showing total runtime and speedup vs number of processors]
Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems

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

- 67.54 seconds to find collision
Benchmarks

Fifth Test: MPI on 6 · 12 Core Systems

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

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

Fifth Test: MPI on 6 · 12 Core Systems

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

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

image source: autoanything
Benchmarks
Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems
Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems
Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems

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

Sixth Test: MPI on 2 · 32 Core Systems

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

Sixth Test: MPI on 2 · 32 Core Systems

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

- 0.96 seconds to find collision
Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: \( \text{md5}(x) \) with \( x \in \{0, 1\}^{24} \)
- 0.96 seconds to find collision
- approx. 4 million strings tried
Benchmarks

Sixth Test: MPI on 2 \cdot 32 Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: $\text{md5}(x)$ 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 \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 deployment of additional node
Benchmarks

Sixth Test: MPI on 2 · 32 Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: $\text{md5}(x)$ 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
Benchmarks
Benchmarks

Seventh Test: MPI on 2 · 32 Core Systems
Benchmarks

Seventh Test: MPI on 2 · 32 Core Systems

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

Seventh Test: MPI on 2 · 32 Core Systems

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

Seventh Test: MPI on 2 · 32 Core Systems

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

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

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

- 97.30 seconds to find collision
Benchmarks

Seventh Test: MPI on 2 · 32 Core Systems

- 2 hosts with 32 Xeon E7-4830 at 2.13GHz
- input: $\text{md5}(x)$ 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 · 32 Core Systems

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

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

image source: autoanything
Explanations and Expectations

image source: ozbinoculars
Explanations and Expectations

OpenMP implementation
Explanations and Expectations

OpenMP implementation

- linear speedup due to few communication
Explanations and Expectations

OpenMP implementation
- linear speedup due to few communication
- suitable for smaller problem sizes
Explanations and Expectations

OpenMP implementation

• linear speedup due to few communication
• suitable for smaller problem sizes
  but: already used CCR’s ”biggest” machine
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
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
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
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
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

\[ x \in \{0, 1\}^{32} \]
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
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

<table>
<thead>
<tr>
<th>compare ( x \in {0, 1}^{32} )</th>
</tr>
</thead>
<tbody>
<tr>
<td># PEs</td>
</tr>
<tr>
<td>OpenMP</td>
</tr>
<tr>
<td>MPI</td>
</tr>
</tbody>
</table>
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

\[ x \in \{0, 1\}^{32} \]

<table>
<thead>
<tr>
<th># PEs</th>
<th>OpenMP</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>152s</td>
<td>174s</td>
</tr>
</tbody>
</table>

compare
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

<table>
<thead>
<tr>
<th># PEs</th>
<th>OpenMP</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>152s</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>174s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>compare $x \in {0, 1}^{32}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEs</td>
</tr>
<tr>
<td>OpenMP</td>
</tr>
<tr>
<td>MPI</td>
</tr>
</tbody>
</table>

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
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)
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
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
Sources and References

