Scalable Asynchronous Contact Mechanics using Charm++

Xiang Ni*, Laxmikant V. Kale* and Rasmus Tamstorf^  
* University of Illinois at Urbana Champaign  
^Walt Disney Animation Studios
Asynchronous Contact Mechanics
Asynchronous Contact Mechanics
Asynchronous Contact Mechanics

• Necessary Guarantees
Asynchronous Contact Mechanics

• Necessary Guarantees
  
  • Safety: no missed collisions
Asynchronous Contact Mechanics

• Necessary Guarantees
  • **Safety**: no missed collisions
  • **Correctness**: follow the laws of physics
Asynchronous Contact Mechanics

• Necessary Guarantees
  • **Safety**: no missed collisions
  • **Correctness**: follow the laws of physics
  • **Progress**: finish in a finite amount of time
Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety**: no missed collisions
  - **Correctness**: follow the laws of physics
  - **Progress**: finish in a finite amount of time

- Problems with other existing algorithms
Asynchronous Contact Mechanics

• Necessary Guarantees
  • **Safety**: no missed collisions
  • **Correctness**: follow the laws of physics
  • **Progress**: finish in a finite amount of time

• Problems with other existing algorithms
  • An object can end up going through itself or another object
Asynchronous Contact Mechanics

• Necessary Guarantees
  • **Safety**: no missed collisions
  • **Correctness**: follow the laws of physics
  • **Progress**: finish in a finite amount of time

• Problems with other existing algorithms
  • An object can end up going through itself or another object
  • Violate physical properties
<table>
<thead>
<tr>
<th>What you want</th>
<th>What you get</th>
</tr>
</thead>
<tbody>
<tr>
<td>pictures from Yi Wang at VT</td>
<td></td>
</tr>
</tbody>
</table>
What you want

What you get

3 pictures from Yi Wang at VT
What you want

What you get

3 pictures from Yi Wang at VT
What you want  
What you get

incorrect handling of collisions

3 pictures from Yi Wang at VT
Parallelization Challenges

- Highly irregular communication pattern
  - *Message driven execution in Charm++*

- Dynamic load imbalancing
  - *Adaptive runtime system*

- Very fine grained computation
  - *Overlapping computation and communication*
Parallelization Challenges

- Highly irregular communication pattern
  - *Message driven execution in Charm++*
- Dynamic load imbalancing
  - *Adaptive runtime system*
- Very fine grained computation
  - *Overlapping computation and communication*
Overall Flow
Overall Flow

Internal Force

Internal Force
Overall Flow

- Internal Force
- Collision Detection
- Internal Force
Overall Flow

- Collision Detection
- Internal Force

Collision Window

Internal Force
Overall Flow

- Internal Force
  - Collision Window
    - Collisions Detected?
    - Detected?
Overall Flow

- Internal Force
  - Collision Window
    - No
    - Proceed to the next window
  - Collision Detection
    - Collisions Detected?
      - No
Overall Flow

- Internal Force
  - Penalty Force

- Collision Detection

- Collisions Detected?
  - Yes
    - Add penalty forces and rollback
  - No
    - Proceed to the next window
Overall Flow

collision response

Internal Force
Penalty Force

Collision Window

Internal Force
Penalty Force
Collision Detection

Yes
Collisions Detected?
Add penalty forces and rollback

No
Proceed to the next window
Collision Detection
Collision Detection

Broad Phase
Collision Detection

Broad Phase

Locally inside each partition, we use a 26-DOP hierarchy to fit the swept volumes of the triangle to detect potential collisions.
Collision Detection

**Broad Phase**

Locally inside each partition, we use a **26-DOP hierarchy** to fit the swept volumes of the triangle to detect potential collisions.

Globally among all the partitions, we fit the trajectory of each triangle to a 3D bounding box and then pass them to the **existing collision detection library** in Charm++.
Collision Detection

**Broad Phase**

Locally inside each partition, we use a **26-DOP hierarchy** to fit the swept volumes of the triangle to detect potential collisions.

Globally among all the partitions, we fit the trajectory of each triangle to a 3D bounding box and then pass them to the **existing collision detection library** in Charm++.
Collision Detection

Narrow Phase

We apply the space-time separating planes method to filter out potential collisions.
Narrow Phase

First Challenge: **Computation Imbalance**
Narrow Phase

First Challenge: **Computation Imbalance**
Narrow Phase

First Challenge: **Computation Imbalance**

![Graph showing time vs. computation and communication phases]
Narrow Phase

First Challenge: **Computation Imbalance**

Time spent on each potential collision pair is *not uniform*
Narrow Phase

First Challenge: **Computation Imbalance**

Time spent on each potential collision pair is **not uniform**

Detection time depends on **trajectory length** of each vertex in the potential pair.
Narrow Phase

First Challenge: **Computation Imbalance**

Time spent on each potential collision pair is **not uniform**

Detection time depends on **trajectory length** of each vertex in the potential pair

A **profiling** based load balancer
Narrow Phase

First Challenge: **Computation Imbalance**

Time spent on each potential collision pair is **not uniform**

Detection time depends on **trajectory length** of each vertex in the potential pair

A **profiling** based load balancer
Narrow Phase

First Challenge: **Computation Imbalance**

Time spent on each potential collision pair is not uniform.

Detection time depends on *trajectory length* of each vertex in the potential pair.

A profiling based load balancer
Narrow Phase

Second Challenge: **Communication Imbalance**

Time (ms)

810ms $\rightarrow$ 150ms

Communication  Computation
Narrow Phase

Second Challenge: **Communication Imbalance**

The more potential collision pairs are spread, the more communication requests.

810ms $\rightarrow$ 150ms
Narrow Phase:
Communication Imbalance

Locality Aware Load Balancer
Narrow Phase: Communication Imbalance

Potential Collisions

- Partition 2 & Partition 3
- Partition 3 & Partition 4
- Partition 5 & Partition 2
Narrow Phase: Communication Imbalance

Locality Aware Load Balancer

Collision Tasks

Partition 2 & Partition 3

Partition 3 & Partition 4

Partition 5 & Partition 2
Narrow Phase: Communication Imbalance

Locality Aware Load Balancer

<table>
<thead>
<tr>
<th>Collision Tasks</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition 2 &amp; Partition 3</td>
<td>Node 3</td>
</tr>
<tr>
<td>Partition 3 &amp; Partition 4</td>
<td>Node 4</td>
</tr>
<tr>
<td>Partition 5 &amp; Partition 2</td>
<td>Node 5</td>
</tr>
<tr>
<td></td>
<td>Node 2</td>
</tr>
</tbody>
</table>
Narrow Phase: Communication Imbalance

Locality Aware Load Balancer

<table>
<thead>
<tr>
<th>Collision Tasks</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition 2 &amp; Partition 3</td>
<td>Node 3</td>
</tr>
<tr>
<td>Partition 3 &amp; Partition 4</td>
<td>Node 4</td>
</tr>
<tr>
<td>Partition 5 &amp; Partition 2</td>
<td>Node 5</td>
</tr>
<tr>
<td></td>
<td>Node 2</td>
</tr>
</tbody>
</table>
Narrow Phase: Communication Imbalance
Overlapping Computation and Communication
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let’s look at the flow on one node
Overlapping Computation and Communication

Let’s look at the flow on one node

L \rightarrow \text{list of potential collision tasks}
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let’s look at the flow on one node

1. Send data request for the external vertices in L
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let’s look at the flow on one node

1. Send data request for the external vertices in \( L \)
2. On receiving message \( M \)
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

1. Send data request for the external vertices in \( L \)
2. On receiving message \( M \)

\[ M \text{.type()} \]
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let’s look at the flow on one node

1. Send data request for the external vertices in $L$
2. On receiving message $M$

Let's look at the flow on one node

$L \leftarrow \text{list of potential collision tasks}$

1. Send data request for the external vertices in $L$
2. On receiving message $M$

Data request

$M \text{.type()}$

sendDataReply()
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let’s look at the flow on one node

1. Send data request for the external vertices in \( L \)
2. On receiving message \( M \)
   - \( M \).type()
   - \( T \leftarrow \) related task
     - if \( T \).ready
       - if \( T \).size > \( \text{THRESHOLD} \)
         - redistribute \( T \) within node
       - else Process \( T \)
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let’s look at the flow on one node

1. Send data request for the external vertices in \( L \)
2. On receiving message \( M \)

\( M \).type()

Data request

sendDataReply()

Data reply

\( T \) ← related task
if \( T \).ready
if \( T \).size > THRESHOLD
redistribute \( T \) within node
else Process \( T \)

Work request

Process subTask(\( M \).start, \( M \).end)
Narrow Phase: Communication Imbalance

Node level data cache

Begin Narrow Phase

Compute Object 1
Compute Object 2
Compute Object 3
Compute Object 4

End Narrow Phase

Node-level software cache.

Is present?

Yes
No.
Send request to remote cache.

Request from remote cache.

Respond.
Narrow Phase

Graph showing the time (s) vs. number of cores for different profiling methods: Naive, Profiling based, Fully optimized, and Linear scaling. The graph indicates linear scaling with increasing number of cores.
Collision Response

Computation Imbalance

![Diagram showing time (ms) and computation response]
Collision Response

Computation Imbalance

Material Force Calculation

Penalty Force Calculation
Collision Response

Computation Imbalance

Material Force Calculation

Penalty Force Calculation
Collision Response

Computation Imbalance

Material Force Calculation

Penalty Force Calculation

4ms → 3.5ms
Collision Response

Importance of partial barrier

Material Force Calculation

Penalty Force Calculation

4ms → 3.5ms
Collision Response

Importance of partial barrier

Material Force Calculation       Penalty Force Calculation

4ms —> 3.5ms
Collision Response

Importance of partial barrier

Material Force Calculation

Penalty Force Calculation

4ms $\rightarrow$ 3.5ms
Collision Response

Importance of partial barrier

Material Force Calculation

Penalty Force Calculation

3.5ms —> 3.2ms
Results

Machines

**Edison**: a Cray XC30, Intel E5-2695@2.4GHz, 12 core Ivy Bridge

**Brickland**: a 4 socket system with Intel E7-4890@2.8GHz, 15 core Ivy Bridge

Examples
Results
Results

(a) Bowline Knot

(b) Reef Knot

(c) Two Cloths Draped

(d) Twister

Charm++ Time (Brickland)  •  Charm++ Time (Edison)  ▲  TBB Time (Brickland)  □
Results

(a) Bowline Knot

(b) Reef Knot

(c) Two Cloths Draped

(d) Twister

Charm++ Time (Brickland)  ✔  Charm++ Time (Edison)  ✔  TBB Time (Brickland)  ✔
Results

(a) Bowline Knot

(b) Reef Knot

(c) Two Cloths Draped

(d) Twister

Charm++ Time (Brickland)  blue
Charm++ Time (Edison) yellow
TBB Time (Brickland) green
Results

(a) Bowline Knot

(b) Reef Knot

(c) Two Cloths Draped

(d) Twister

Charm++ Time (Brickland)  
Charm++ Time (Edison)  
TBB Time (Brickland)
Long Twister
Long Twister
Long Twister

![Graph of simulated time versus time. The graph shows a peak at around 15s, with values ranging from 0 to 5 in the y-axis and from 0 to 50 in the x-axis. Notable points include: 5s, 10s, 15s, and 25s.]
Long Twister

(a) 5s

(b) 10s

(c) 15s

(d) 25s

Time per window:  
- Force calculation
- Collision detection

Number of Cores

Time [s]
Conclusion

• Strong scaling to 384 cores on Edison

• More than 10x speedup compared to the TBB shared memory results

• Charm++ is well-suited for dynamic irregular applications like ACM

• Message-driven feature helps the overlapping of communication and computation