Case study: Parallel orbit enumeration

Max Neunhöffer

HPCGAP workshop 19–23 August 2013

(joint work with Christopher Brown, Kevin Hammond,
Vladimir Janjic, Steve Linton and Hans-Wolfgang Loidl)

Problem (Orbit enumeration)

Let $a : X \times G \to X$ and $x_0 \in X$. Determine the smallest subset $\mathcal{O} \subseteq X$, such that $x_0 \in \mathcal{O}$ and: for all $x \in \mathcal{O}$ and all $g \in G$ we have $a(x, g) \in \mathcal{O}$.

Problem (Orbit enumeration)

Let $a : X \times G \to X$ and $x_0 \in X$. Determine the smallest subset $\mathcal{O} \subseteq X$, such that $x_0 \in \mathcal{O}$ and: for all $x \in \mathcal{O}$ and all $g \in G$ we have $a(x, g) \in \mathcal{O}$. $\mathcal{O}$ is called the G-orbit of x_0, denoted by $x_0 G$.

Problem (Orbit enumeration)

Let $a : X \times G \to X$ *and* $x_0 \in X$*. Determine the smallest subset* $\mathcal{O} \subseteq X$*, such that* $x_0 \in \mathcal{O}$ *and: for all* $x \in \mathcal{O}$ *and all* $g \in G$ *we have* $a(x, g) \in \mathcal{O}$*.* $\mathcal{O}$ *is called the G-orbit of* x_0*, denoted by* x_0G*. We write* $x \cdot g$ *for* $a(x, g)$*.*

Problem (Orbit enumeration)

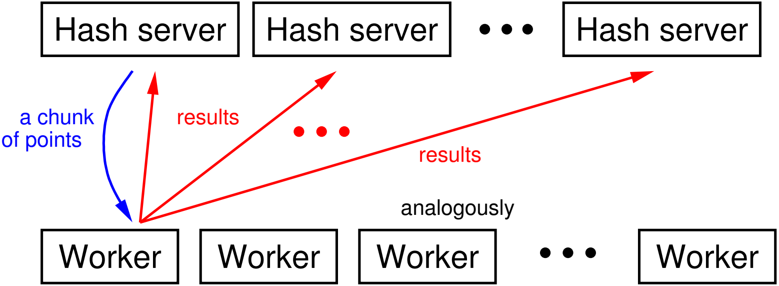
Let $a : X \times G \to X$ *and* $x_0 \in X$*. Determine the smallest subset* $\mathcal{O} \subseteq X$*, such that* $x_0 \in \mathcal{O}$ *and: for all* $x \in \mathcal{O}$ *and all* $g \in G$ *we have* $a(x, g) \in \mathcal{O}$*.*
$\mathcal{O}$ *is called the G-orbit of* x_0*, denoted by* x_0G*. We write* $x \cdot g$ *for* $a(x, g)$*.*
Often, G is a generating system of a (semi-)group.

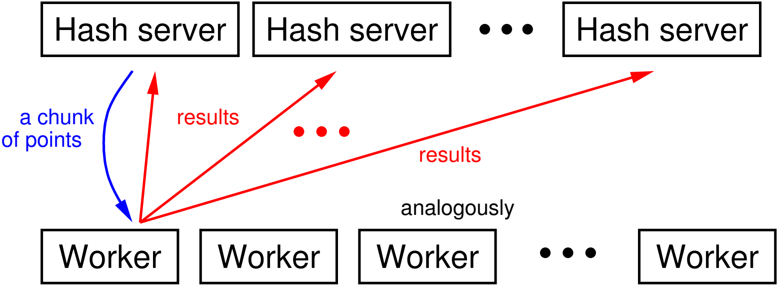
Problem (Orbit enumeration)

Let $a : X \times G \to X$ and $x_0 \in X$. Determine the smallest subset $\mathcal{O} \subseteq X$, such that $x_0 \in \mathcal{O}$ and: for all $x \in \mathcal{O}$ and all $g \in G$ we have $a(x, g) \in \mathcal{O}$. $\mathcal{O}$ is called the G-orbit of x_0, denoted by x_0G. We write $x \cdot g$ for $a(x, g)$. Often, G is a generating system of a (semi-)group.

Basic Orbit Algorithm

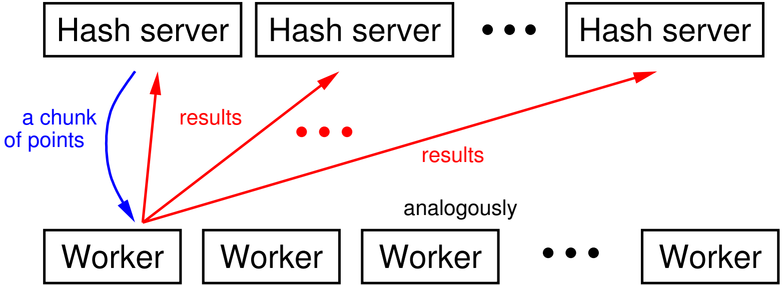
```
Input: $x_0 \in X, g_1, g_2, \ldots, g_k : X \to X$
$T := \{x_0\}$   (a hash table);   $O := [x_0]$   (a list);   $i := 1$
while $i \leq \mathrm{Length}(O)$ do
    for $j$ from 1 to $k$ do
        $y := O[i] \cdot g_j$
        if $y \notin T$ then
            Add $y$ to $T$
            Add $y$ to the end of $O$
    $i := i + 1$
return $O$   (containing the orbit of $x_0$)
```


Hash server
Hash server
• • •
Hash server
a chunk
of points
results
• • •
results
analogously
Worker
Worker
Worker
• • •
Worker



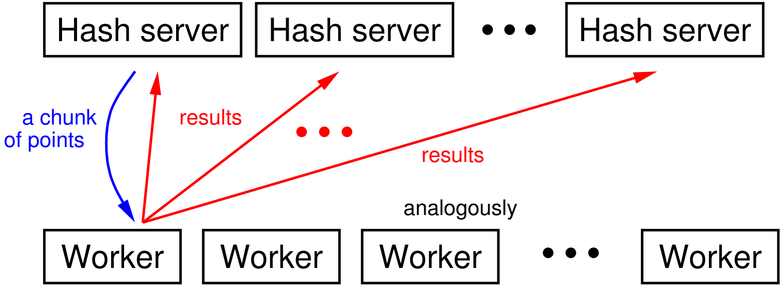
A **worker**

- gets a chunk of points from some hash server,



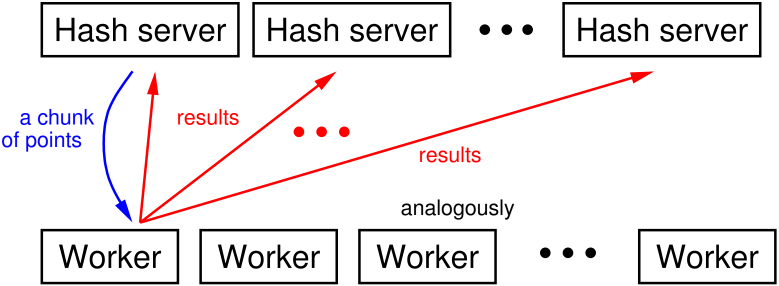
A **worker**

- gets a chunk of points from some hash server,
- applies all generators to all points in the chunk,



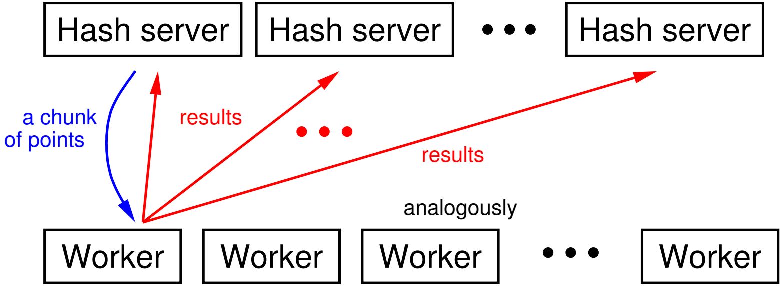
A **worker**

- gets a chunk of points from some hash server,
- applies all generators to all points in the chunk,
- and sends all results to the responsible hash server.



A **worker**

- gets a chunk of points from some hash server,
- applies all generators to all points in the chunk,
- and sends all results to the responsible hash server.
- A distribution function regulates who is responsible.

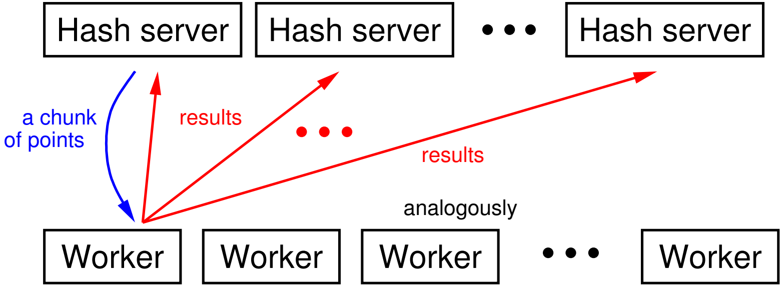


A **worker**

- gets a chunk of points from some hash server,
- applies all generators to all points in the chunk,
- and sends all results to the responsible hash server.
- A distribution function regulates who is responsible.

A **hash server**

- stores and recognises points, and



A **worker**

- gets a chunk of points from some hash server,
- applies all generators to all points in the chunk,
- and sends all results to the responsible hash server.
- A distribution function regulates who is responsible.

A **hash server**

- stores and recognises points, and
- keeps track of work to do.

A worker

Input:

- the set G and the action function $a : X \times G \to X$,
- the number h of hash servers and
- a distribution hash function $f : X \to \{1, \ldots, h\}$

```
while TRUE do
    get a chunk $C$ of points
    $R$ := a list of length $h$ of empty lists
    for all $x \in C$ do
        for all $g \in G$ do
            $y := x \cdot g$
            append $y$ to $R[f(y)]$
    for all $j \in \{1, \ldots, h\}$ do
        schedule sending $R[j]$ to hash server $j$
```

A hash server

```
Input: a chunk size $s$
initialise a hash table $T$ and a work queue $Q$
while TRUE do
    get a chunk $C$ of points (usually from a worker)
    for all $x \in C$ do
        if $x \notin T$ then
            add $x$ to $T$ and append it to $Q$
            if at least $s$ points in $Q$ are unscheduled then
                schedule a chunk of size $s$ points
    if there are unscheduled points in $Q$ then
        schedule a chunk of size $< s$ points
```

All points are immutable and read only objects.

All points are immutable and read only objects.

We have ignored the termination condition here.

All points are immutable and read only objects.

We have ignored the termination condition here.

The same basic model can be used in shared memory and in distributed memory.

All points are immutable and read only objects.

We have ignored the termination condition here.

The same basic model can be used in shared memory and in distributed memory.

In the shared memory implementation we use channels to communicate chunks of points.

All points are immutable and read only objects.

We have ignored the termination condition here.

The same basic model can be used in shared memory and in distributed memory.

In the shared memory implementation we use channels to communicate chunks of points.

For more details see

`hpcgap/demo/parorbit/parallelorbit2.g`

All points are immutable and read only objects.

We have ignored the termination condition here.

The same basic model can be used in shared memory and in distributed memory.

In the shared memory implementation we use channels to communicate chunks of points.

For more details see

`hpcgap/demo/parorbit/parallelorbit2.g`

Vladimir will talk about the distributed memory implementation.

We estimate the amount of communication:

- Every point has to be sent to one worker.

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results,

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action,

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action,
  then every point in the orbit is found equally many times.

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action,
  then every point in the orbit is found equally many times.
- $\Longrightarrow$ Need to transfer $(|G| + 1) \cdot |\mathcal{O}|$ points.

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action, then every point in the orbit is found equally many times.
- $\Longrightarrow$ Need to transfer $(|G| + 1) \cdot |\mathcal{O}|$ points.
- We assume that the distribution function works well.

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action, then every point in the orbit is found equally many times.
- $\Longrightarrow$ Need to transfer $(|G|+1)\cdot|\mathcal{O}|$ points.
- We assume that the distribution function works well.

We use queues everywhere to avoid latency:

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action, then every point in the orbit is found equally many times.
- $\Longrightarrow$ Need to transfer $(|G| + 1) \cdot |\mathcal{O}|$ points.
- We assume that the distribution function works well.

We use queues everywhere to avoid latency:

- Each hash server has an input queue.
- There is a global work queue to send work to the workers.
- We use one more channel for termination and result collecting.

We estimate the amount of communication:

- Every point has to be sent to one worker.
- Every point produces $|G|$ results, which have to be sent back to some hash server.
- If G generates a group and a is a group action, then every point in the orbit is found equally many times.
- $\Longrightarrow$ Need to transfer $(|G|+1) \cdot |\mathcal{O}|$ points.
- We assume that the distribution function works well.

We use queues everywhere to avoid latency:

- Each hash server has an input queue.
- There is a global work queue to send work to the workers.
- We use one more channel for termination and result collecting.

In general: Never use blocking calls for communication!

Problem (Filling the queues)

The main difficulty is to fill the queues!

Problem (Filling the queues)

The main difficulty is to fill the queues!

The whole process starts by feeding x_0 to some hash server.

Problem (Filling the queues)

The main difficulty is to fill the queues!

The whole process starts by feeding x_0 to some hash server.

At first only few workers have work.

Problem (Filling the queues)

The main difficulty is to fill the queues!

The whole process starts by feeding x_0 to some hash server.

At first only few workers have work.

However, in the beginning every point produces up to $|G|$ new points.

Problem (Filling the queues)

The main difficulty is to fill the queues!

The whole process starts by feeding x_0 to some hash server.

At first only few workers have work.

However, in the beginning every point produces up to $|G|$ new points.

If the growth of the number of unprocessed points is not fast enough, the workers starve.

Problem (Filling the queues)

The main difficulty is to fill the queues!

The whole process starts by feeding x_0 to some hash server.

At first only few workers have work.

However, in the beginning every point produces up to $|G|$ new points.

If the growth of the number of unprocessed points is not fast enough, the workers starve. If we avoid this problem, we get:

Problem (Filling the queues)

The main difficulty is to fill the queues!

The whole process starts by feeding x_0 to some hash server.

At first only few workers have work.

However, in the beginning every point produces up to $|G|$ new points.

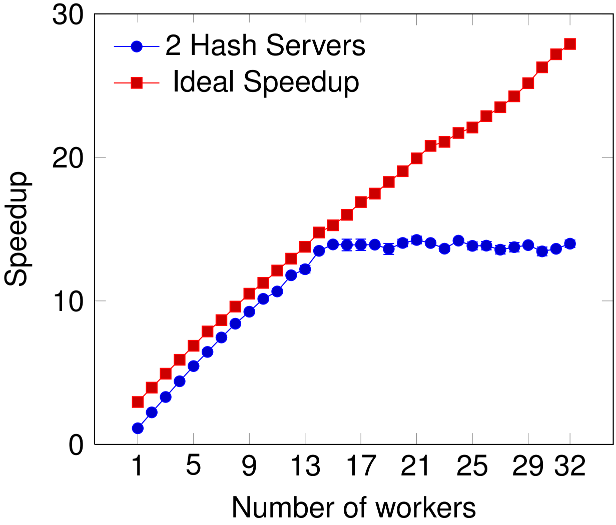
If the growth of the number of unprocessed points is not fast enough, the workers starve. If we avoid this problem, we get:

Theorem (A priori runtime estimate)

Let w be the number of workers and h be the number of hash servers. Then the runtime of our algorithm is approximately

$$\max\left\{\frac{|G|\cdot|\mathcal{O}|}{wA}, \frac{|G|\cdot|\mathcal{O}|}{hL}\right\},$$

where A is the number of ACT *operations a worker can do per sec. and L is the number of* LOOKUP *operations a hash server can do per sec.*


2 Hash Servers
Ideal Speedup
Speedup
30
20
10
0
1
5
9
13
17
21
25
29
32
Number of workers