# Paralellizing large search in BOINC: a case study

Wenjie Fang, Université Paris Diderot
in co-operation with yoyo@home

BOINC Workshop 2013, Grenoble, France

the wrapper platform
Yoyo @ home
for natural science

## Brute-force search

Sometimes we just don't have good algorithms.

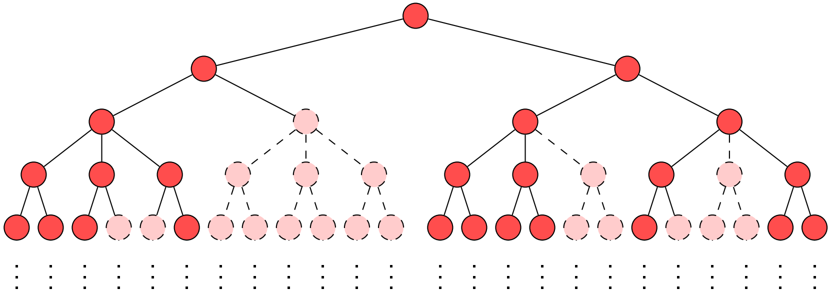
- Satisfaction problem and constraint programming
- Combinatorial optimization
- Artificial intelligence
- Conjecture verification
- etc...

**Quotation**

When in doubt, use brute force.

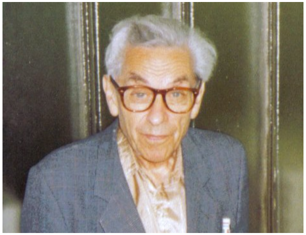
– *Ken Thompson*

## Backtracking



## Our case: Odd Weird Search

A **weird** number is a notion related to proper divisors. It is unknown whether an odd weird number exists, and the great mathematician **Paul Erdös** offered a prize for it.



$$S(n) = \{k \ / \ k < n, k|n\}. \sum_{k \in S(n)} k > n \wedge \forall S' \subset S(n), \sum_{k \in S'} k \neq n.$$

We want to search for an odd weird number.

## Our case: Odd Weird Search

To check if a number is weird, we need to obtain its proper divisors.

- **Naïve way**
  For each odd number, we **factorize** it, calculate its proper divisors, solve a subset sum problem to determine if it is weird.
  Factorization very expensive, but easy to partition.
- **Backtracking**
  We construct (or search) directly the factorization, and backtrack whenever possible. The rest is the same.
  Much faster, but difficult to partition.

Of course we go the difficult but faster way.

## Difficulties

Although much faster, backtracking is difficult to paralellize.

- No **good and simple** estimate of search volume, only **rough and simple** ones
- Irregularity of subtrees

In the BOINC context, it is even harder. We want to meet the following demand to please our volunteers:

- Reasonable and consistent runtime for **every** workunit ($< 12$h)
- Reasonable progress bar
- Easy workunit generation

None of them is easy.

## Previous attempts

There are already some projects that parallelize their search of similar flavor.

- Rectilinear Crossing Number (with occasional extremely huge workunits)
- NQueens@home (smooth search space)
- SAT@home (expensive Monte Carlo estimation)
- etc...

## Our solution

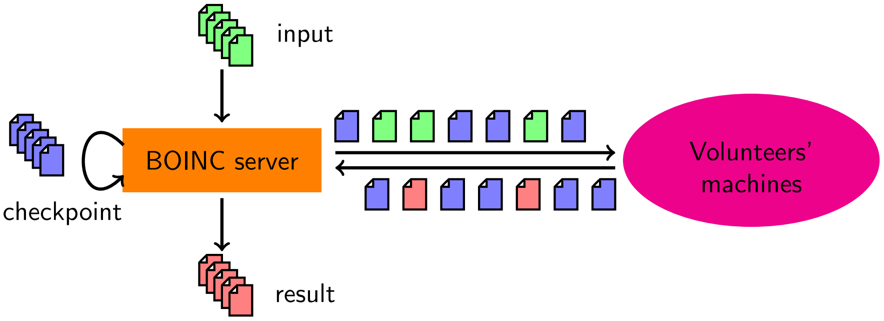
No good and simple estimate, so we just use rough and simple ones!
Rough estimate + Irregular subtrees ⇒ workunits of varying size

### Our trick

Just force every workunit to **stop after some time** and then **send back its checkpoint**.
We then **recycle** checkpoints sent back as workunits.

We wrote a mechanism to **roughly** estimate running time by counting operations in expensive functions.
This estimate can be off by $20\%$, but still consistent, and we have a control.
And we have a progress bar for **free**!

## What does it look like?



For simplicity, three types of file share the same format.

## Issues

At the end, throughput will drop due to low “liquidity”.

- Shorten deadline to increase “liquidity”
- Send more initial replica to shorten waiting time
- Compute locally when not many are left

To assure correctness, we do a quorum 2.
Our scheme only works when we only care about the search as a whole, and when the checkpoint is not large.

## Conclusion

Our scheme can be used to parallelize a large class of search with a rough search volume estimate. In fact, an upperbound would work.
In fact, it works as a potential solution to large workunits, if latency is not important.
Future work:

- Better recycle strategy
  - Estimation of search volume of recycled workunits
  - Priority of different recycled workunits
- Automatic control of “liquidity”
- Quantitative analysis?

Thanks for your attention!