FAST REGEX PARSING ON GPUS

David Lehavi & Sagi Schein
HP Labs Israel

PLEDGE

- Although this is a lecture about GPUs, this is the only piece of cool graphics you’ll see:

(picture from Assassin’s Creed Revelations)

REGULAR EXPRESSIONS (RE)

- One of the most common programming languages.
  - Standard syntax.
  - Platform independent.

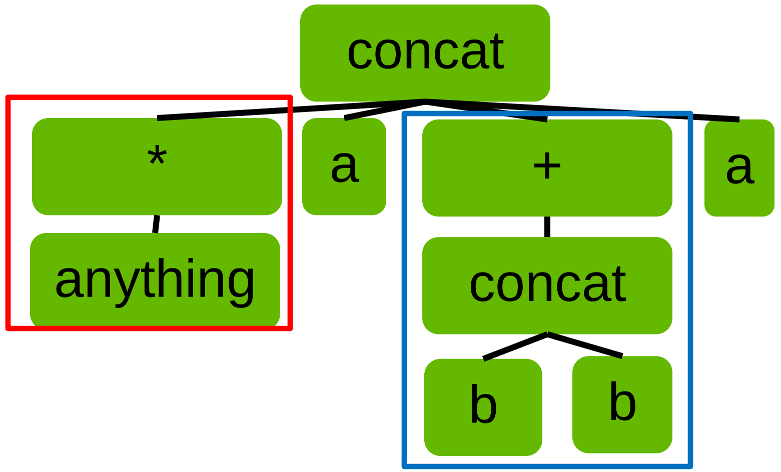
.*a(bb)+a

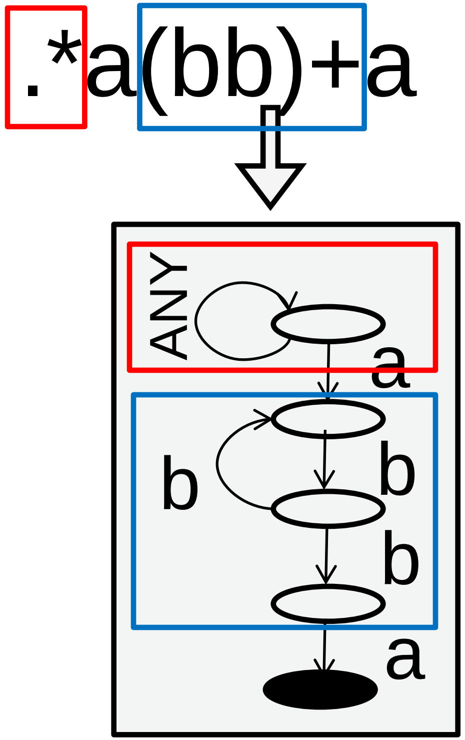
- Commonly used for “text crunching applications”:
  - Security applications.
  - Network monitoring.
  - Database queries (SQL, Google’s BigTable, …).
- Common scenario: many strings & REs.
  - **Pitfall: high run-time variance.**
  - **Throughput ⇒ run time are critical.**
  - Latency is not always critical.

REGEXS AN AUTOMATON

Ken Thompson’s approach

- Transform the regex to an automaton.
  - Parse to an abstract syntax tree.
  - “Translate” to an automaton

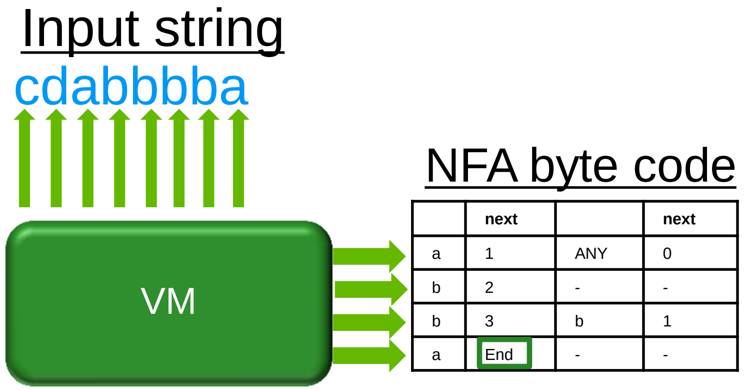




WHEN AUTOMATONS BECOME DATA

Using virtual machines

- Virtual machines are not always VMware, or JVM, or the likes of them.
- Virtual machine “runs” on
  - the string (data)
  - and a table representing the automaton (bytecode).
- Looks ***only*** at the next character.
- Maintaine “**Front**” of active states.



	next		**next**
a	1	ANY	0
b	2	-	-
b	3	b	1
a	End	-	-

THE BOUNDARY BETWEEN VM AND CODE

Universal Turing machine:

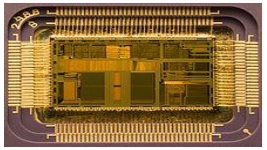
```
current = states[current, tape[place]]

place += increment[current]
```

Silly machine, smart byte code, slow, no branches ⇒ Can be parallelized assuming cache and shared memory ⇒ GPU friendly.

Modern CPU:

Smart machine, silly byte code, fast, many (HW implemented) branches ⇒ not GPU friendly.



Any hardware has it's own sweetspot

In GPUs arithmetic is cheap, brunches are expansive

BRUNCHES VS INDICATORS

Instead of (bad - serialized divergent brunches)

```
if (s1[n] == s2[m] )
  a[n,m] += match
else
  a[n,m] += mismatch;
```

Use an indicator (good - cheap arithmetic):

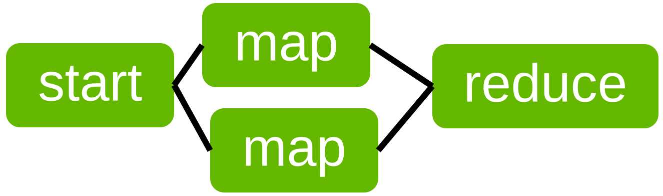
```
i = (s1[n] == s2[m]);
a[n,m] += match + i *(mismatch - match);
```

- Sometimes better on CPU (no brunch prediction), always better on GPUs and SIMD (e.g. SSE).
- Exponential build-up , no precedence in evaluation.

TASK PARALLELISM ON GPUS

MAP REDUCE

- The abstract concept, not Hadoop.
- Similar length tasks.
- Tasks distributed by **map**
- Data is collected and "merged" by **reduce**.



HUNGRY PUPPIES

- Varied length tasks
- No mapping/reduction is needed.
- Each "puppy" takes more "food" when it is done.

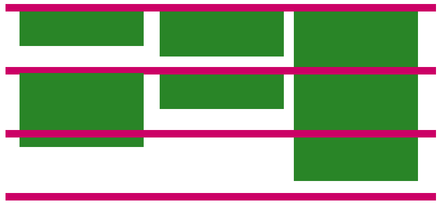
WHAT IF…..

- More tasks than threads.
- map-reduce friendly where reduction is:
  - order free
  - almost conflict free.
- Tasks:
  - inherently iterative
  - very different lengths.

BALANCING MAP-REDUCE

Overcoming high runtime variance: from data to task parallelism

```
while (has task) {
  map free threads
  while (work_load < max and
    not finished) {
      perform one iteration
      work_load ++
  }
  reduce finished threads
}
```



- **Far better than map reduce due to varying task lengths**
- Ideally max computed dynamically.
- Reduce is trivial for regexes.

Rescheduling is done without quitting the kernel code, all the data is still in shared memory

Memory access is not coalesced, but it is to shared memory/cache, so it doesn't matter

Similar ideas were explored in distributed systems – different problems/solutions

FEATURES AND BENCHMARK

- Ready to use C++ library (HP internal)
  - All standard features: POSIX wildcards, ranges, sub- matches, Giga char strings, greedy/non-greedy operators, Boyer Moore searches. Support for multiple cards.

- Benchmark data: two hundred strings (6K char each), 200 starting points each, 36 REs, 5 Boyer-Moore (times are in sec).
  - PCRE is considered to be the best regex engine.
  - Latency is about 0.1 sec for search 0.2 sec for match.
  - Memory upload total latency (with an ancient bus): 1-1.5 sec.

- Naïve extrapolation: 12U systems assuming full parallelization – optimizing **compute density**
  - Six HP Proliant SL390s, each is 4U/2 box, dual 6-core Xeons, 8 Nvidia Tesla M2090.
  - Two HP blade system c3000, each is a 6U, 8 half blades of two 6-core Xeons.
  - Compare 120 3.46Ghz Xeon5690 to 48 Tesla M2090.

	1 thread PCRE 2.67Ghz Xeon	RegexGPU Quadro 4000
1.44M RE	269.5	27.5 search 57.5 match
200K BM	1.6	0.032

performance	**/vol**	**/Watt**
RE match	4	1
RE search	8	2
Boyer Moor	50	10