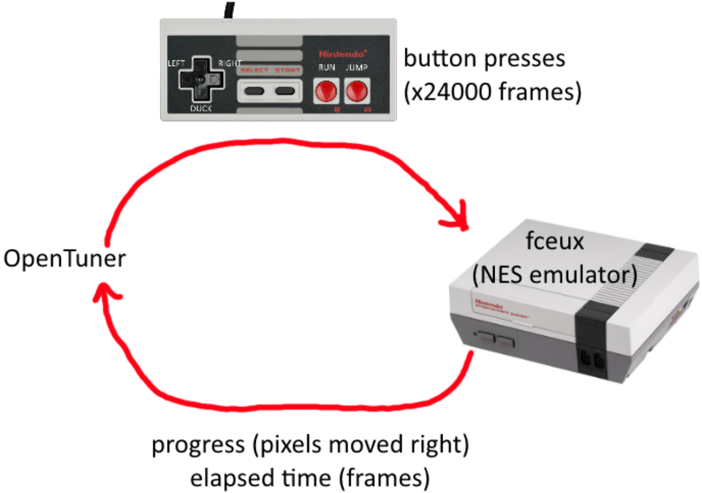
Super Mario Bros. problem

Input space

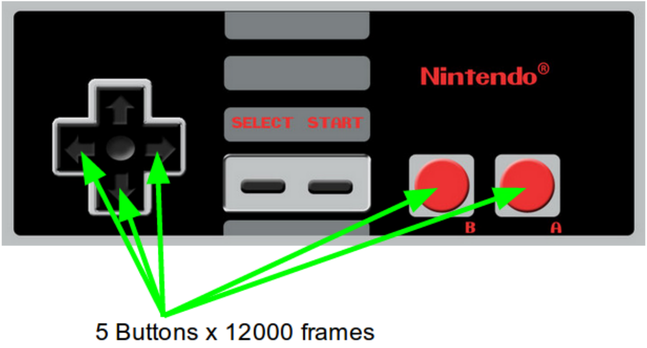
- 5 buttons per frame
- 24000 frames
- $5^{24000} \approx 1.9 \times 10^{16775}$ possible input sequences

Exhaustive search won't work here.

Tuning process

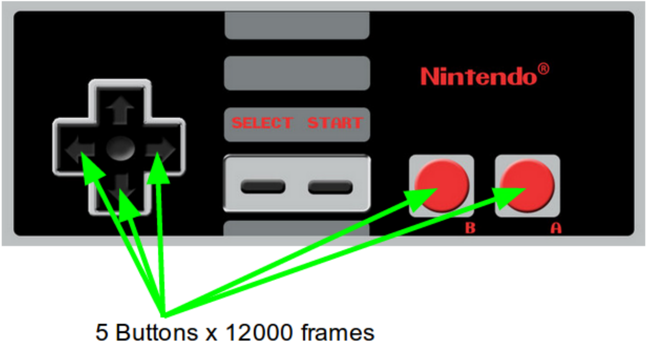


Naive Representation



[1] http://youtu.be/nyYdq1jJQrw

Naive Representation



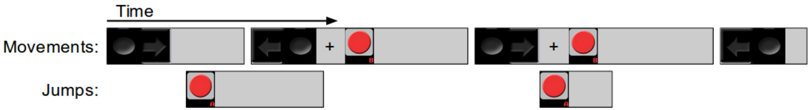
- Bad, because most configurations make no sense.
- Just mashing random buttons.
- Doesn't work at all (Video [1]).

[1] http://youtu.be/nyYdq1jJQrw

Better Representation

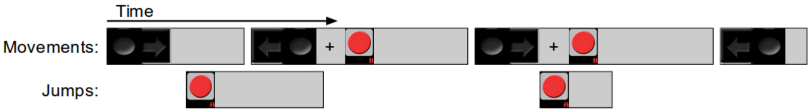
- Movements (list):
  - Direction (left, right, run left, or run right)
  - Duration (frames)

Better Representation



- Movements (list):
  - Direction (left, right, run left, or run right)
  - Duration (frames)
- Jumps (list):
  - Start frame
  - Duration (frames)

Better Representation

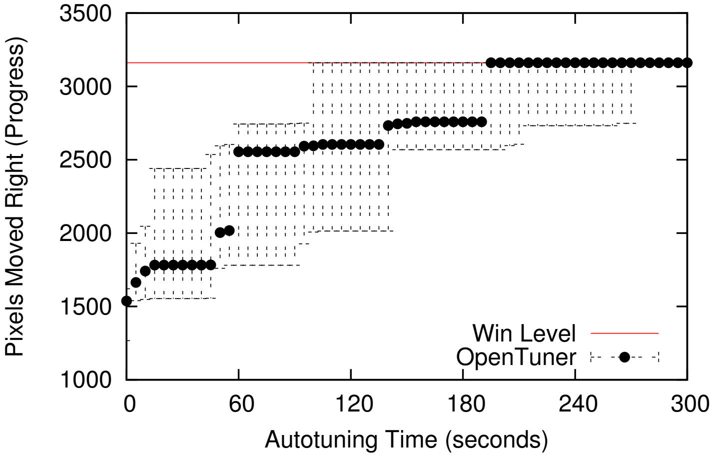


- Movements (list):
  - Direction (left, right, run left, or run right)
  - Duration (frames)
- Jumps (list):
  - Start frame
  - Duration (frames)

Choosing the right representation is critical

- Search space size 10^{6328}
- Winning run found in 13641 ($\approx 10^4$) attempts
- Under 5 minutes of training time

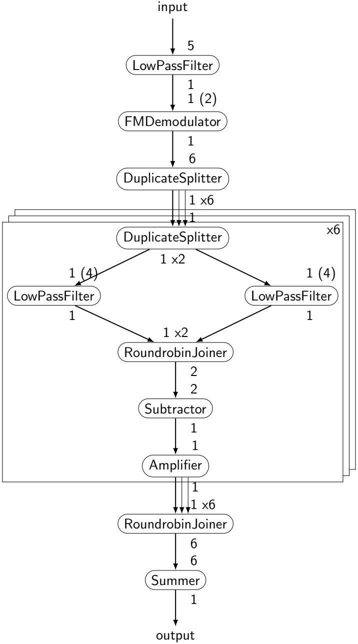
Super Mario Bros Results



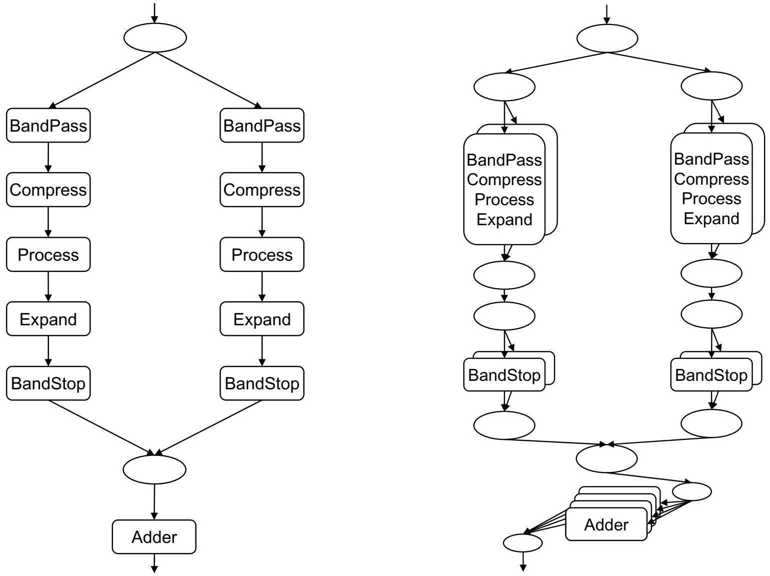
StreamJIT

Synchronous dataflow programs are graphs of (mostly) stateless workers with statically-known data rates.

Using the data rates, the compiler can compute a schedule of worker executions, fuse workers and introduce buffers to remove synchronization, then choose a combination of data, task and pipeline parallelism to fit the machine.



Fusion, data-parallel fission and splitter/joiner removal



Autotuning

StreamJIT delegates its optimization decisions to OpenTuner, which decides

- an overall schedule multiplier (to amortize synchronization)
- whether to fuse workers
- whether to remove splitters and joiners
- buffer implementations
- how to allocate fused groups to cores

Autotuning work allocation

Equal distribution across all cores is usually the best, but we need to load-balance around stateful workers.

- Bitset per worker, one bit per core: exponentially hard to get equal distribution (all bits set).
- Array of floats summing to 1.0, one float per core: allows load-balancing, but equal distribution is even harder.

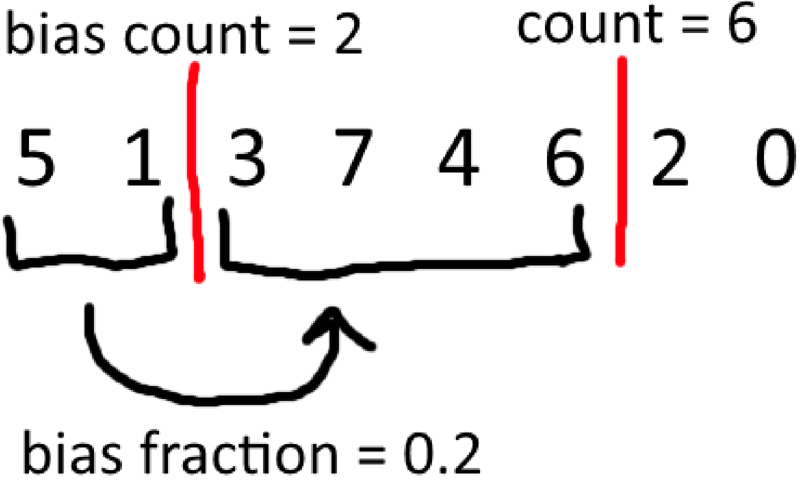
Autotuning work allocation

Equal distribution across all cores is usually the best, but we need to load-balance around stateful workers.

- Bitset per worker, one bit per core: exponentially hard to get equal distribution (all bits set).
- Array of floats summing to 1.0, one float per core: allows load-balancing, but equal distribution is even harder.
- Permutation of cores, total count, bias count and bias fraction: equal division across cores, biased for load balancing.

Bias fraction work allocation

Use the first *count* cores of the permutation, moving *fraction* of the work from the first *bias count* cores.



Doesn't cover all possibilities, but covers the good ones.

Custom techniques

StreamJIT uses custom techniques that force the obvious defaults.

Other techniques make some good and some bad changes:

↑-↓--↑-↓↑↑-↓

Custom techniques will then force some of the bad changes back:

↑----↑-↓↑↑--

Bandit will learn to stop using the custom techniques when they stop working or for unusual graphs where the obvious defaults are bad.