ATOS: an Auto-Tuning Optimization System

A. Moynault, H. Knochel, R. Duraffort,
D. Ferranti, C. Robine, C. Vincent, C. Guillon,
F. de Ferrière

2013-12-06

life.augmented

Auto Tuning Optimization System

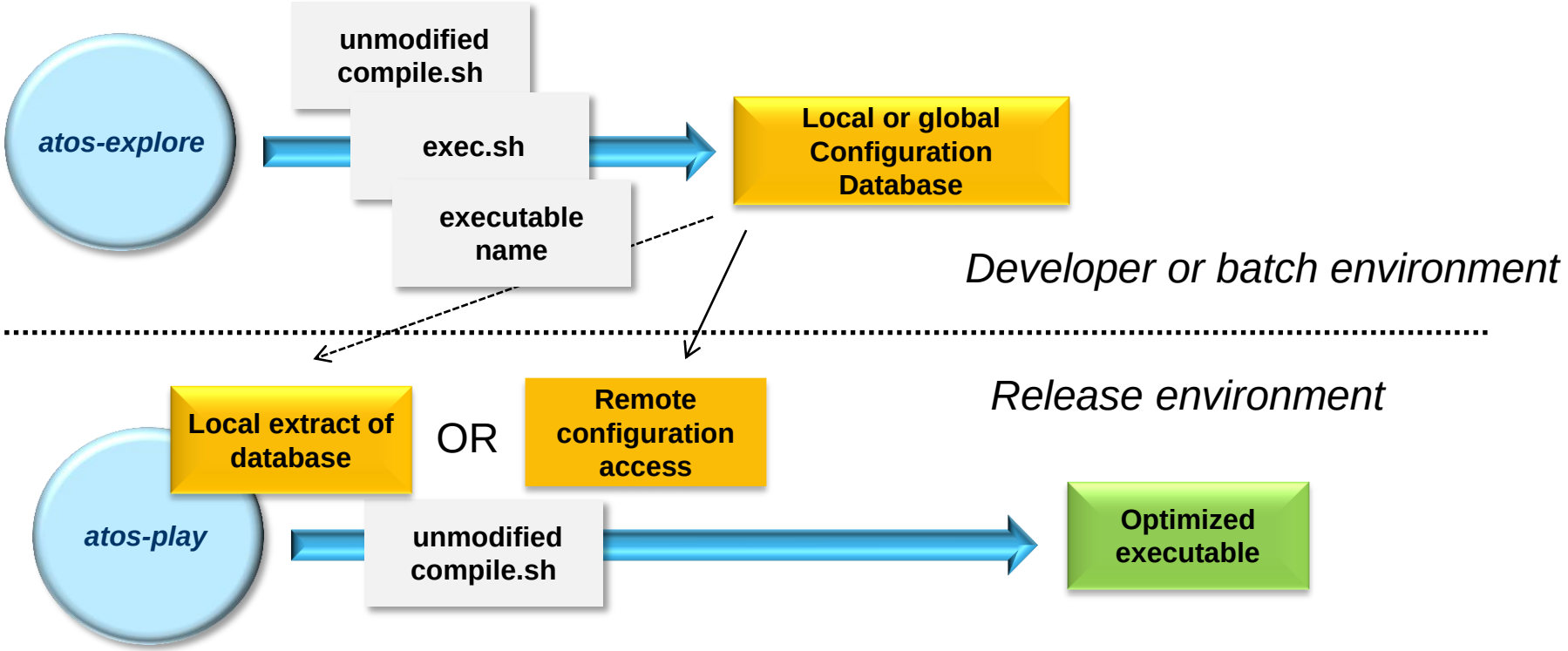
ATOS stands for:

"Auto Tuning Optimization System"

It is based on iterative compilation optimization

- Functional requirements
  - Find automatically a best C/C++ compiler configuration for some given objectives
  - Explore on top of any build system with a given set of representative use cases
- Application
  - Best performance / size tradeoffs search for:
    - a given set of executables/libraries/kernel modules
    - a given set of benchmarking use cases
  - Requires a "representative" use case

ATOS high level usage



Unmodified compile.sh: actually any build system command:

- make all
- rpmbuild
- build.sh
- . . .

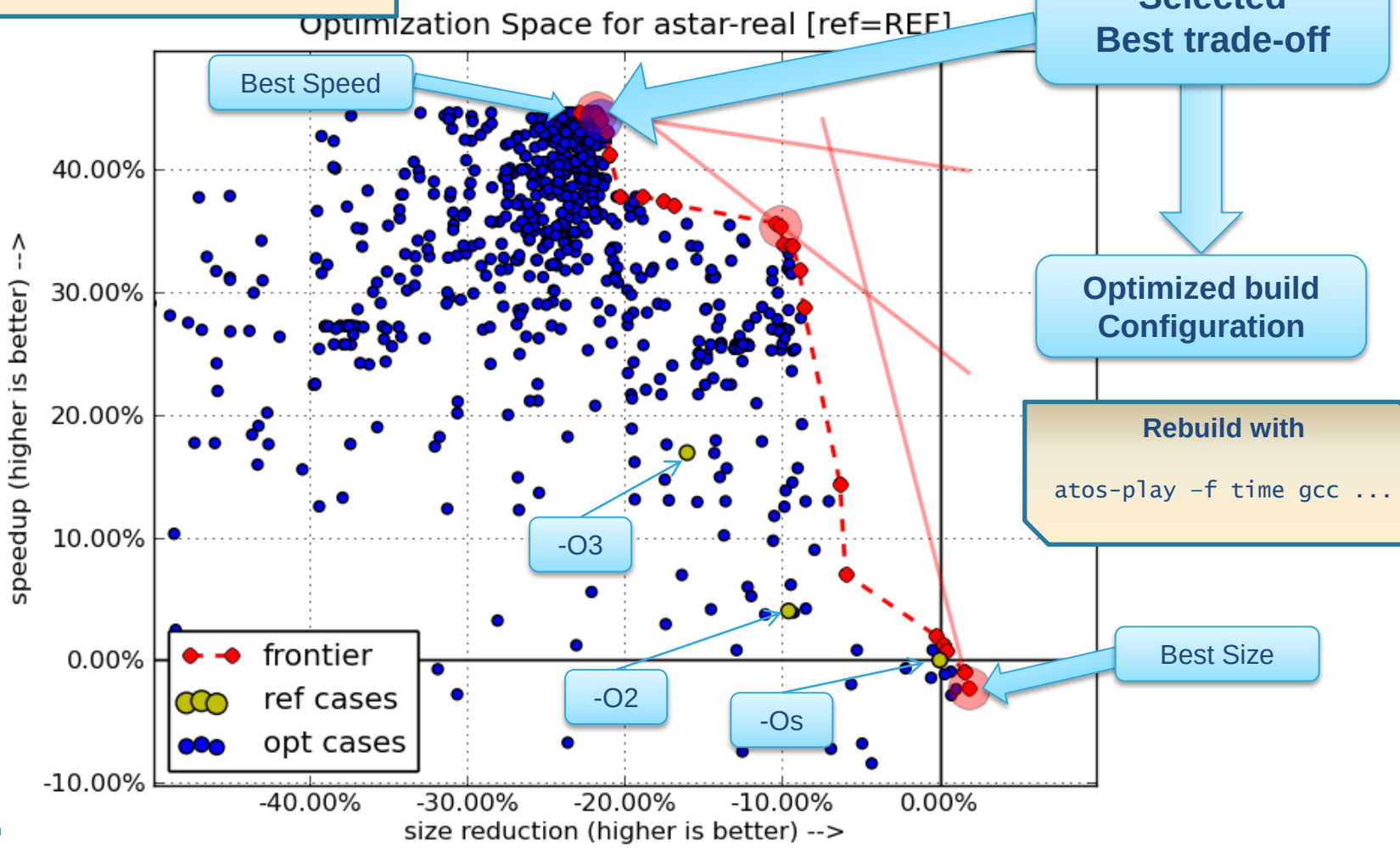
exec.sh: actually a script running the application

- make run
- direct or remote execution
- run.sh
- emulated or on board

ATOS Exploration and Replay Example

Explore with

```
atos-explore -b build.sh -r run.sh
```



life.augmented

ATOS Features

- Simple command line interface
- Easy to use on top of a build system
  - Only requires a build and run scripts
  - Uses PRoot to intercept calls to the compiler and change command line options
  - Limited to Linux
- Easy to get performance figure
  - By default ATOS uses user time
  - Or the run script can return its own figure
  - ATOS can also compute an average on multiple runs
    - Better results in cases execution time has some variation
- Support of GCC 4.5/4.6/4.7/4.8 / LLVM 3.1 / RVCT

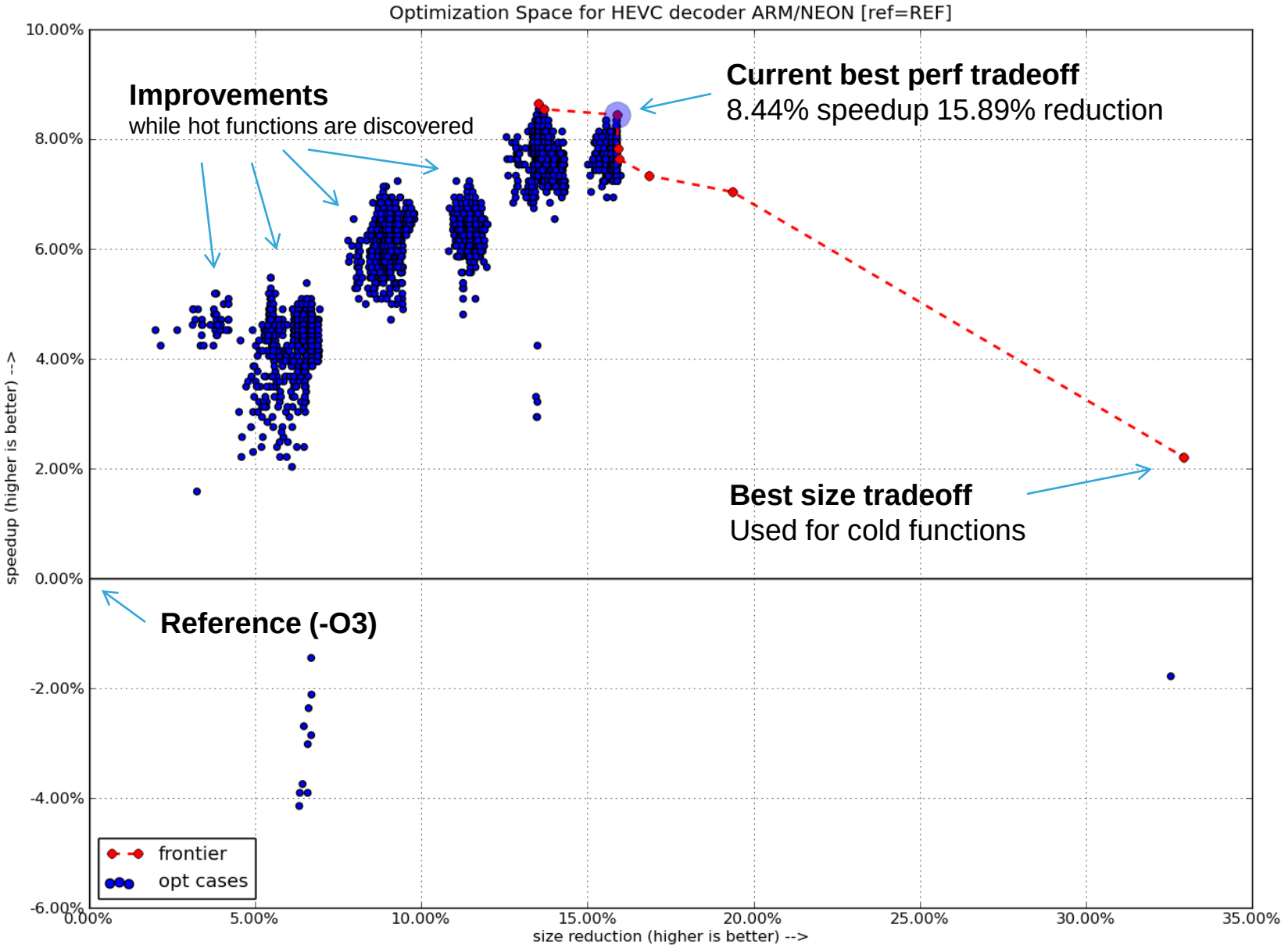
ATOS Features (Cont'd)

- Support for 'perf' and 'oprofile‘ tools
- Provide seamless profiling feedback and link time optimizations
- Native or cross build and run
- Can execute build and run scripts in parallel
  - Compilation on farms
    - archiving & sandboxing provided by CARE & PRoot
  - Execution on farms
    - sandboxing & functional emulation provided by PRoot and QEMU
- Web portal to gather and share results

ATOS Features (Cont'd)

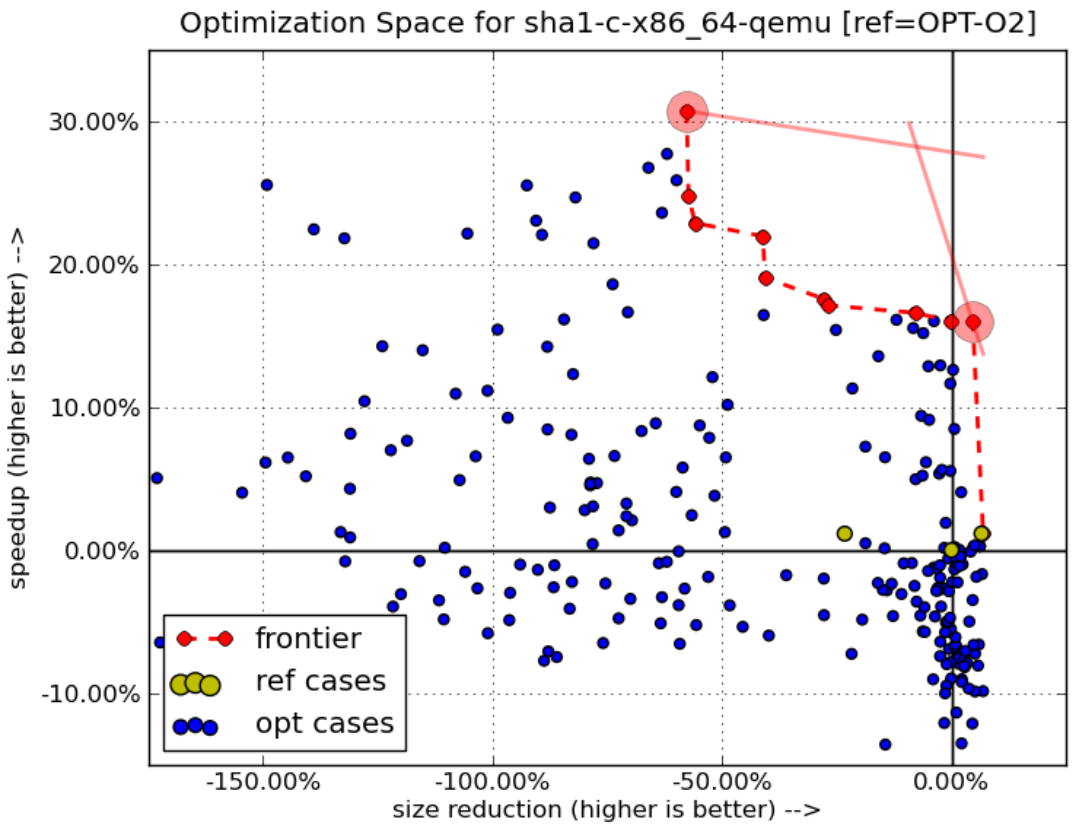
- Whole executable optimization sequences
  - Find the best sequence of flags for the whole application
- File by file exploration of optimization sequences
  - Use profiling to make a hot/cold partitioning of the files
  - Hot files are optimized for speed, explorations are performed on each file
  - Cold files are optimized for size
- Function by function exploration with provided GCC plugins
  - Use profiling to make hot/cold partitioning of functions
  - Hot functions are optimized for speed, explorations are performed one function at a time
  - Cold functions are optimized for size
  - A GCC plugin is used
    - to read an ACF (Application Configuration File) file that describes options and parameters for a list of functions
    - To pass these options and parameters to GCC when a function is compiled

ATOS function-by-function exploration



A parallel exploration

- An exploration with parallel build and run
  - Parallel build and run
  - On a farm of 50 CPUs
  - Exploration with 250 random sequences
  - Accelerated 3x Real time : 4’



Exploration of the option space

- The exploration space is huge
  - On GCC 4.6.2, ATOS can tune 115 -f options and 61 --param options
    - But ATOS has flag's dependencies to avoid generating meaningless sequences
  - An exploration to tune an application typically consists of a few hundred to several thousand option sequence

- We experimented with various techniques for the exploration of the option space
  - Random
    - Generate random sequences of options
  - Staged
    - The option space is divided into three sets : inline, loop, optim
    - Search for best option sequence is performed successively in each set
  - Genetic
    - Compiler flags are *genes*
    - A sequence of compiler flags defines an *individual*
    - A set of compiler flags sequences is a *generation* of individuals
    - The algorithm applies genetic transformations to individuals so as to improve a fitness function on individuals over generations

A genetic algorithm for ATOS

- Implemented *evolution* and *mutation* genetic operators
  - No experiment with cross-over
  - Values tuned by experiments confirmed what is found in the literature
    - Probability for a sequence to mutate instead of evolve: 0.3
    - For a sequence's evolution, probability for a flag to evolve: 0.3
    - For a sequence's mutation, probability for a flag to mutate: 0.3
    - Only one best sequence is kept at the end of a generation to seed the next generation
- Improved the genetic algorithm with static flag ranking
  - We expect that only a small fraction of the 115 gcc options has a significant, good or bad, impact on the performance or code size
  - Compute a ranking of flags from a few benchmarks
  - Use this ranking for explorations on other benchmarks
    - The ranking can be used on random, staged and genetic explorations
  - Best flags will have more probability to be selected
  - Worst flags will have less probability to be selected
  - Only on –f/-fno- flags, not on –param options

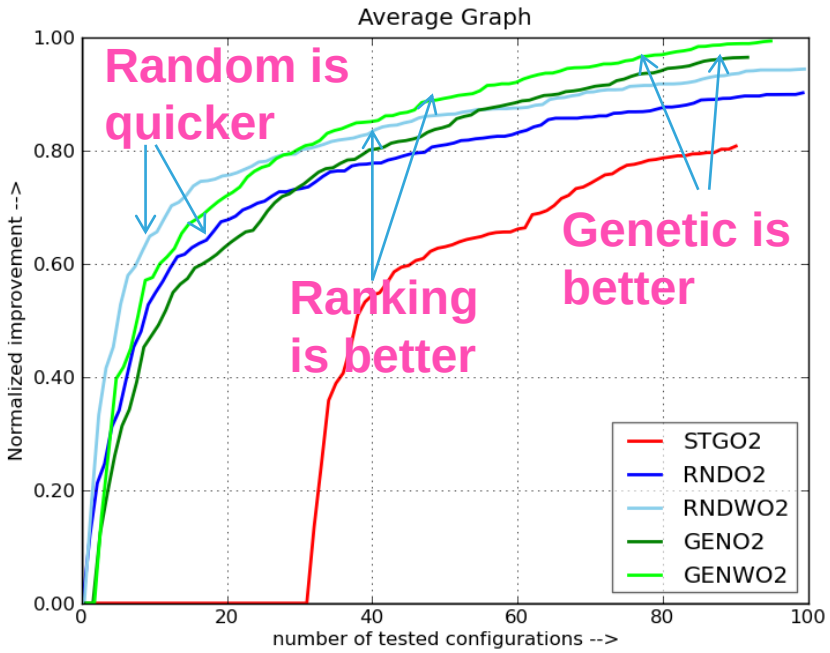
Experimentation

- Performed explorations on 10 embedded application cores of a few hundred lines
- Compared random, staged and genetic explorations
  - Each exploration is ran 20 times with a different random seed to reduce random bias
  - Small explorations
    - Only 100 sequences, or 10 generations of 10 sequences
    - But still requires about one week to complete.
- Flag ranking was used on random and genetic explorations
  - Flag ranking was computed on three other benchmarks
    - bzip2, sha1, spec2006/astar
  - 80% -finline-functions, -fearly-inlining, -finline-small-functions, -funroll-all-loops, -ftree-loop-optimize, -fivopts, -funroll-loops, -fguess-branch-probability, -ftree-sra
  - 20% -ftree-ch, -ftree-pre, -fselective-scheduling2, -ftree-vrp, -fforward-propagate, -fschedule-insns

Exploration Results

- Normalized the results of each benchmark to compute an average

- Staged explorations failed to find good combinations with two or more options



- Genetic algorithm improves over random exploration
  - Random exploration give better results on the first sequences
  - Genetic algorithm becomes better than random after 3 generations

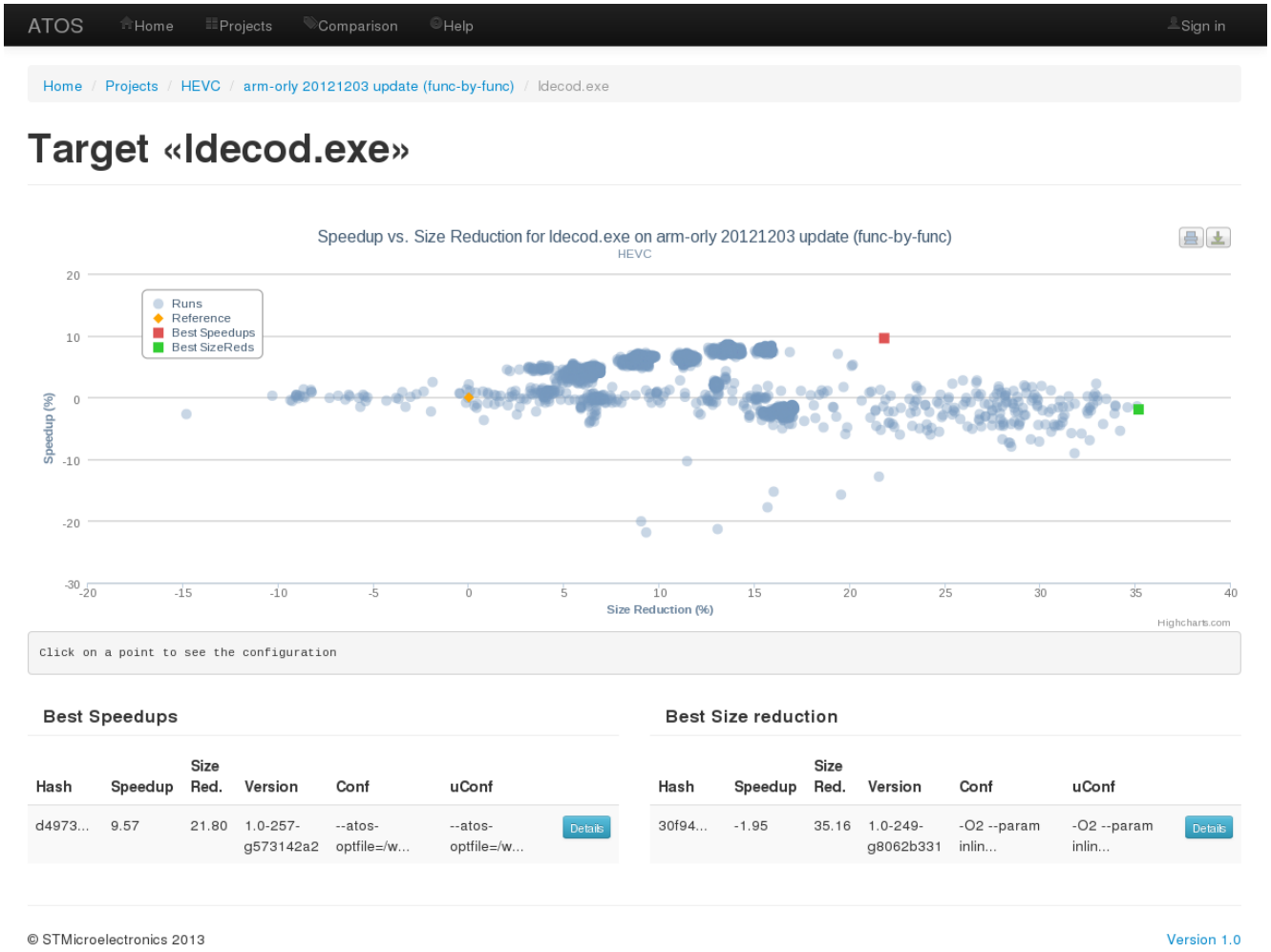
- Flag ranking performs well

ATOS optimization results

- JPEG ST40 / HDK7108 results
  - 26.39% speedup / 13.37% size increase
- ZLIB ST40 / HDK7108 results
  - 12.54% speedup / 1.41% size increase
- Stagecraft ST40 / HDK7108 results
  - 5-28% speedup (30 benchs) / 14% size reduction
- HEVC ARMv7 / Orly results
  - 9.22% speedup / 21.21% size reduction
- SPEC2000 ARMv7 / U9540 results
  - 18.7% speedup SPECINT / 10.2% speedup SPECFP / size increase n/a

ATOS Portal

- Exploration results can be archived on an ATOS web portal



Conclusion

- Good results so far, with interesting speedups on real applications
  - ATOS has been used to optimize the HEVC application and standard Linux libraries
- ATOS may also prove very interesting on not well tuned compilers for recent architectures
- Will continue to work on the genetic algorithm
  - Add dynamic flag ranking
- But ATOS is not publicly available yet