COLE: Compiler Optimization Level Exploration

Kenneth Hoste and Lieven Eeckhout
Ghent University, Belgium

CGO-2008, Boston (MA)
April 8th, 2008

UNIVERSITEIT GENT

Cost/benefit trade-off

Compiling a program means trading off between various objectives

- compilation time, code quality, code size, ...

Constructing a good set of optimization levels (-Ox) is hard

- conflicting objectives, unknown effect of some optimizations, complex interactions, ...

Trying to please everyone

Constructing optimization levels is currently based on heuristics and experience, e.g. in GCC:

- ➡ low compilation cost, doesn't tamper with debugging ⇒ -O1
- ➡ may increase code size, likely better code quality ⇒ -O2
- ➡ high compilation cost, maybe yield even better code quality ⇒ -O3

Each optimization level allows trading off various objectives

-O1: optimize, and be quick about it

-O2: take your time, give it your best shot

-O3: I'm feeling lucky, and have lots of time

Optimization for the masses

How to obtain a set of optimizations that would serve well as a standard optimization level?

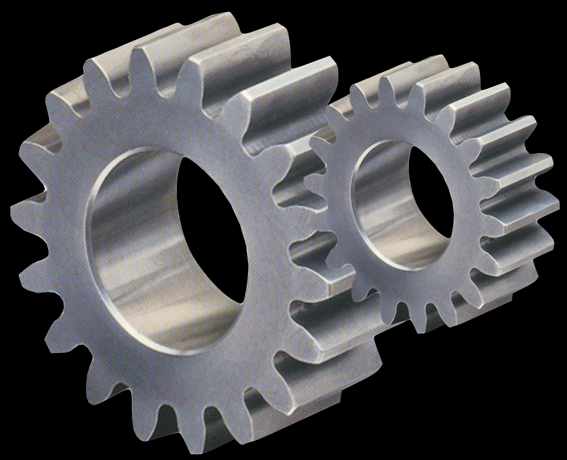
Related techniques show impressive results...

iterative optimization
but:
- for a single target metric
- for a single application

dynamic compilation (JIT)
but:
- for a single target metric
- manual, or based on heuristics

COLE
Compiler Optimization Level Exploration

allows to automatically construct Pareto-optimal compiler optimization levels

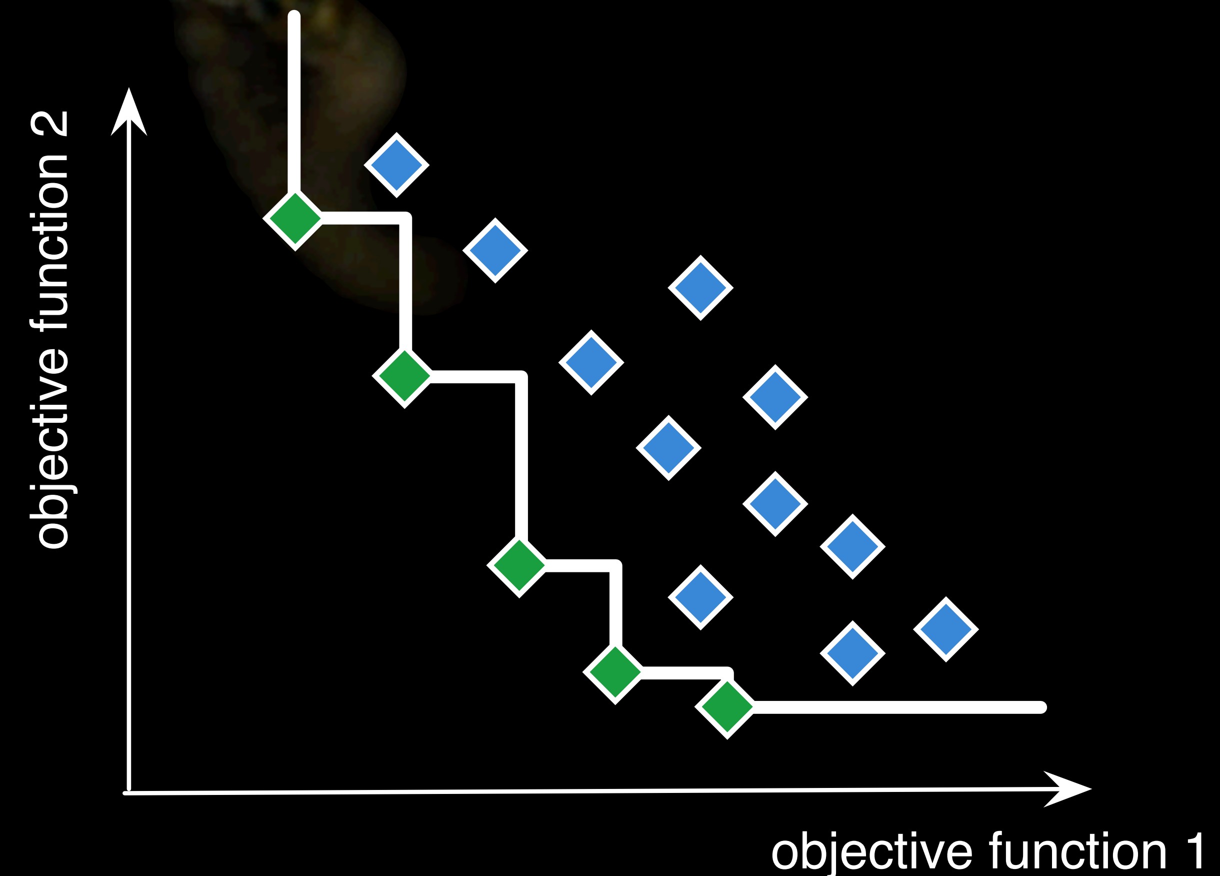


- ➡ multi-objective evolutionary search algorithm
- ➡ Pareto-optimal over a given set of benchmarks
- ➡ outperforms random searching and heuristics
- ➡ completely transparent to compiler, metrics, benchmarks, ...
- ➡ detailed analysis yields interesting insights

Try and beat me

Pareto-optimal optimization level

no other candidate optimization level achieves a better score for all objective functions

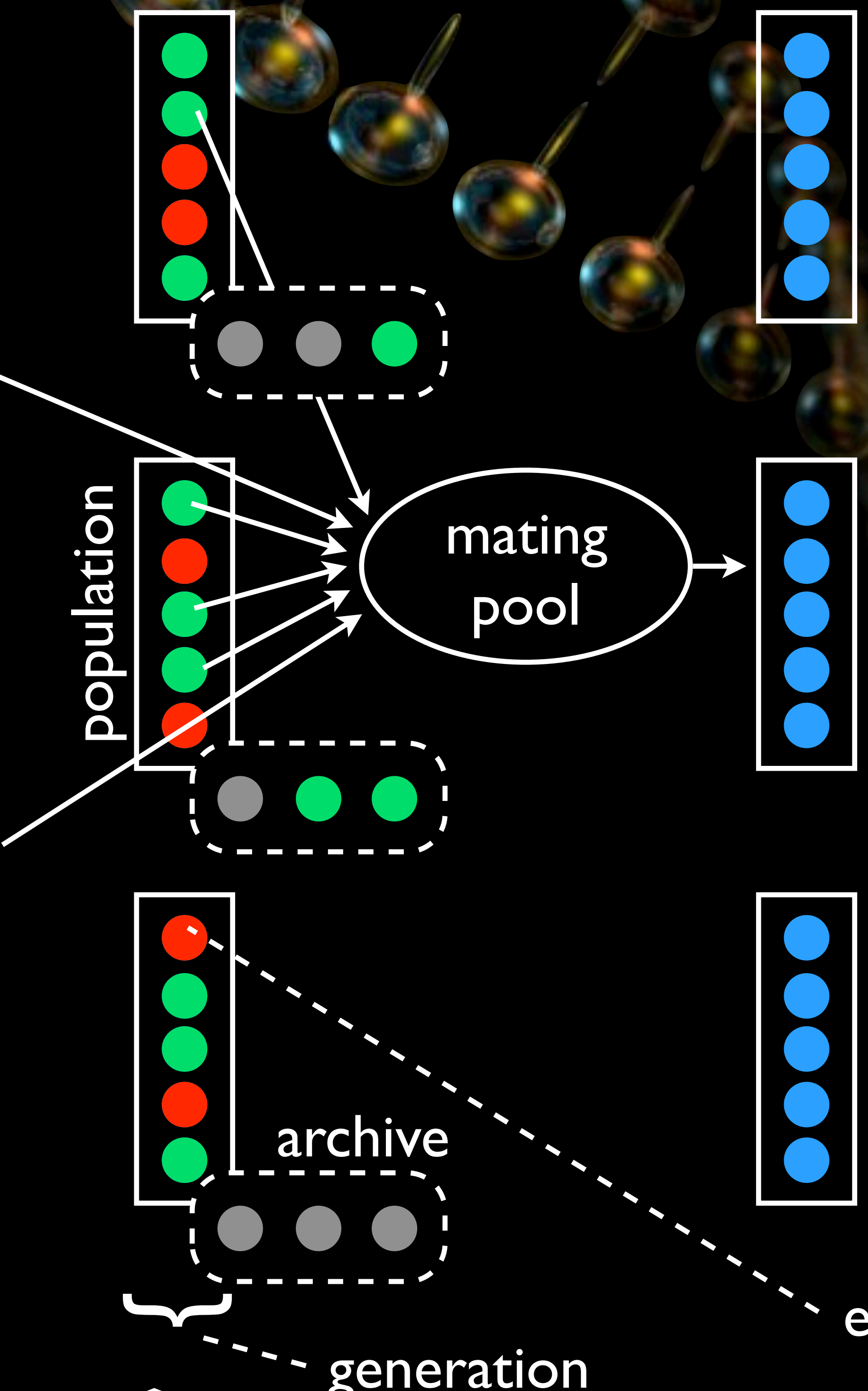


Survival of the fittest

multi-objective searching based on SPEA2 (Zitzler et al.)

- ➡ evolutionary search strategy
- ➡ each entity presents a candidate optimization level, consisting of a combination of optimizations
- ➡ multiple populations of entities are evolved, following survival of the fittest
- ➡ the end result is a set of Pareto-optimal optimization levels

Survival of the fittest



- ➡ initial generation: random entities (and -Ox)
- ➡ each entity is evaluated, score is based on dominance relationship with others
- ➡ mating pool is filled with dominating entities (and the best from previous populations)
- ➡ new entities are generated by combining and mutating ones from the mating pool
- ➡ convergence = no improvement for 3 subsequent generations
- ➡ 3 populations, 20 entities, archive of 10

entity: candidate optimization level
(combination of optimizations)

Putting it to the test

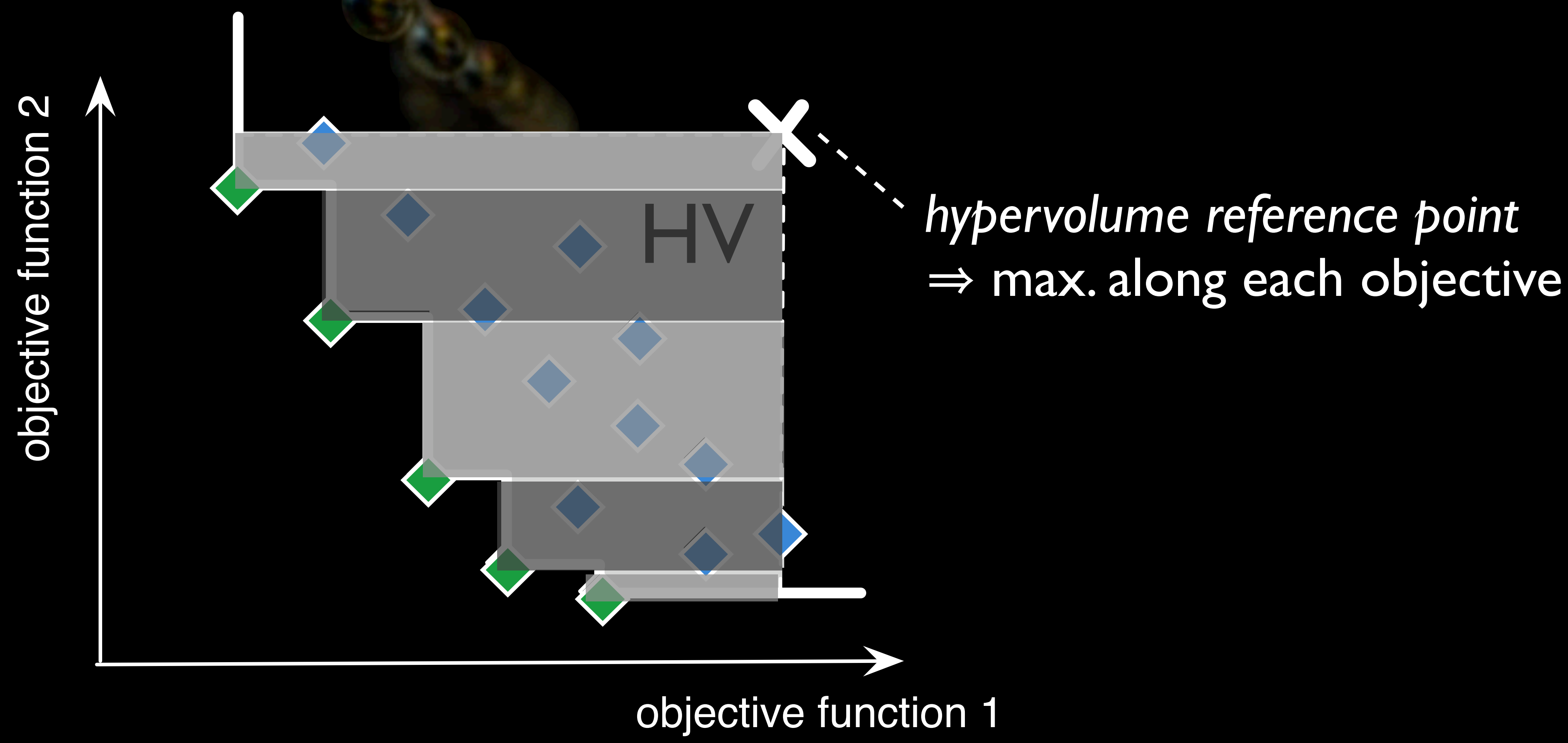
Experimental setup:

- *GCC 4.1.2*, using all *60* optimizations enabled at -O3
- the compilation level is set at either -O1, -O2 or -Os (but without using the corresponding optimizations)
- *objectives*: compilation time and execution time
- *benchmarks*: SPEC CPU2000 (train)
- *platform*: Linux / Intel Pentium 4 Prescott 3.0 GHz
- running COLE to convergence took *50 days* (~4,200 entities tested)

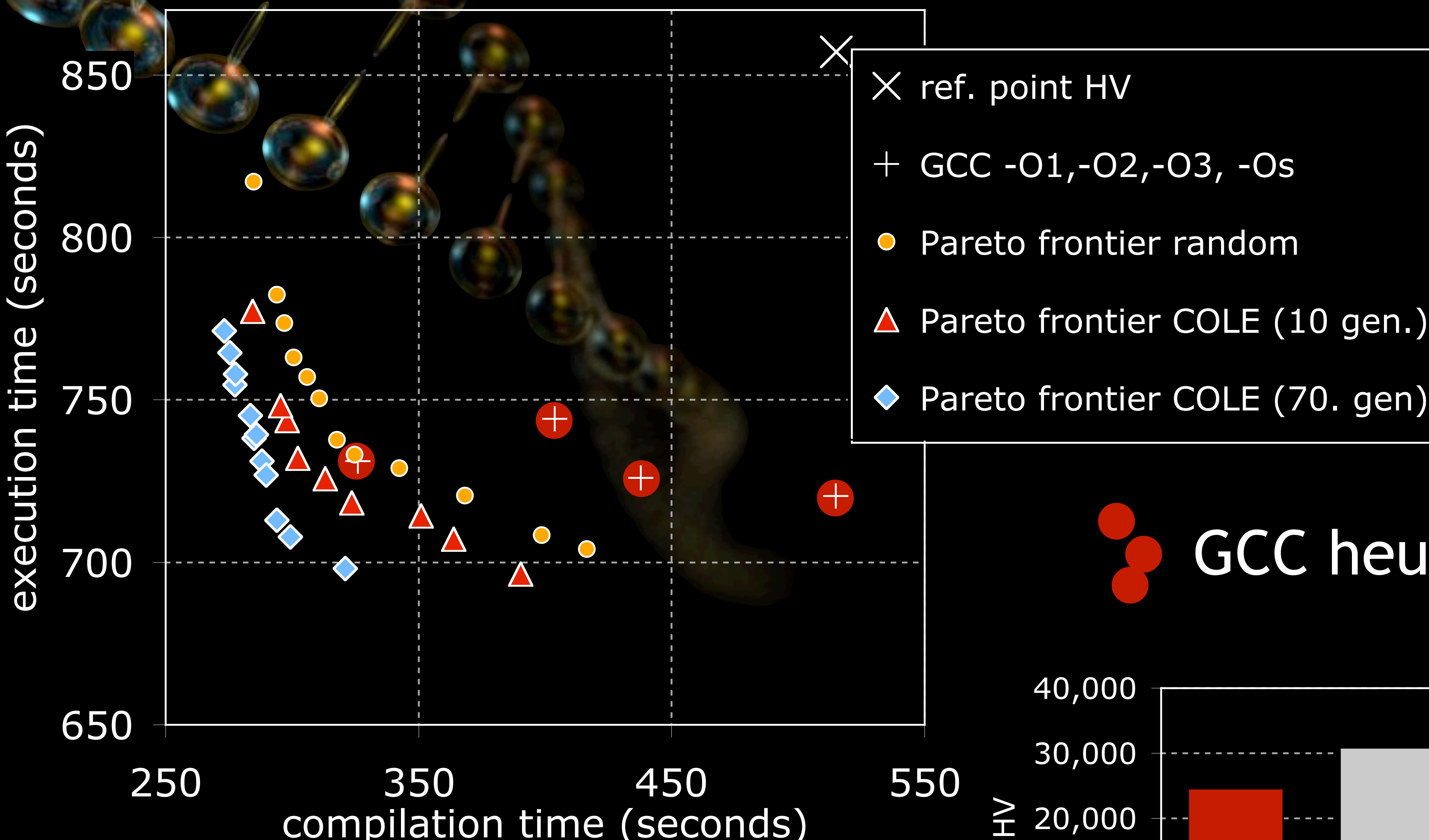
  but is *embarrassingly* parallel: speedup of 60x by evaluating entire generation at once on parallel machines

Quantifying multi-objective superiority

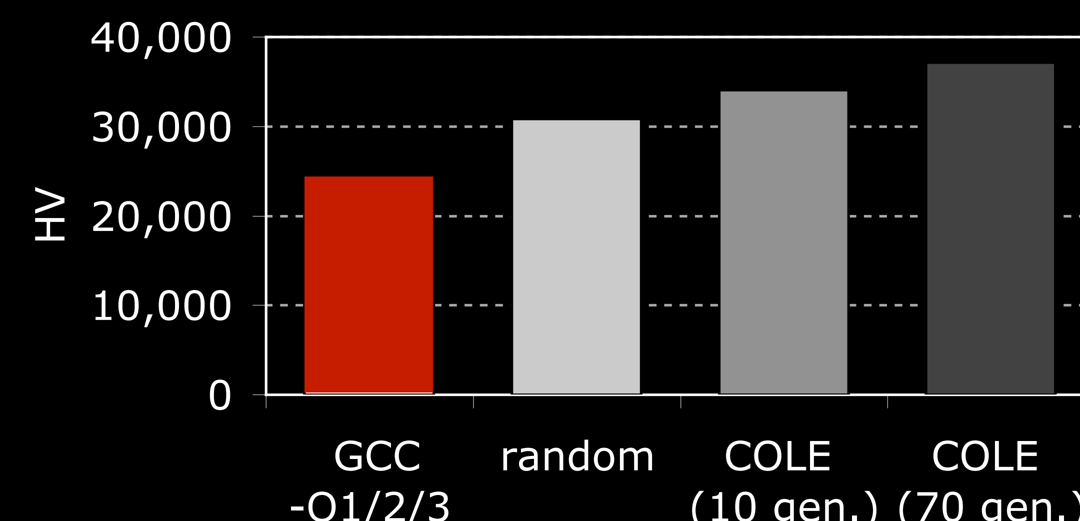
Quality of a set of Pareto-optimal entities is quantified using the hypervolume (HV) metric



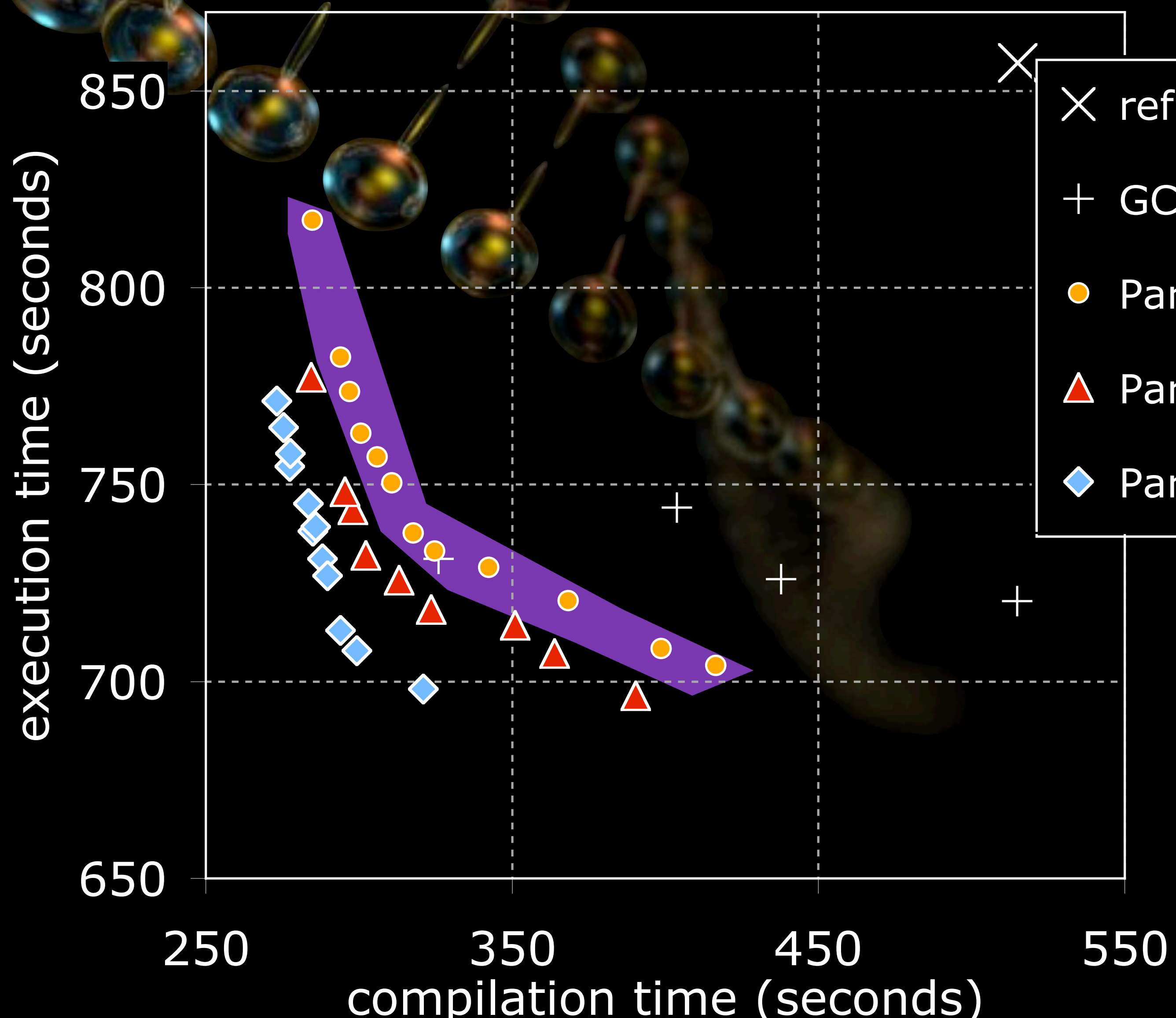
And the winner is...



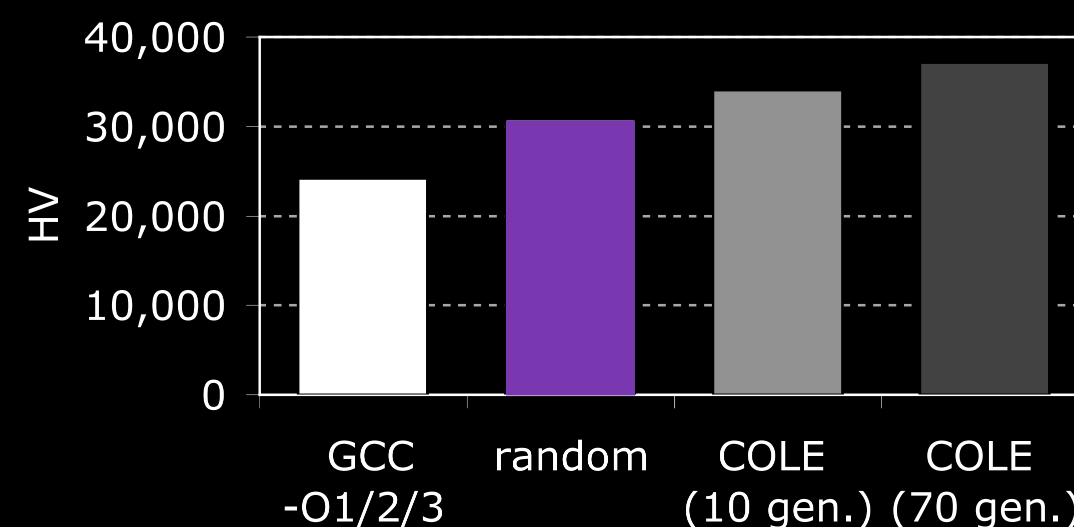
GCC heuristics



And the winner is...

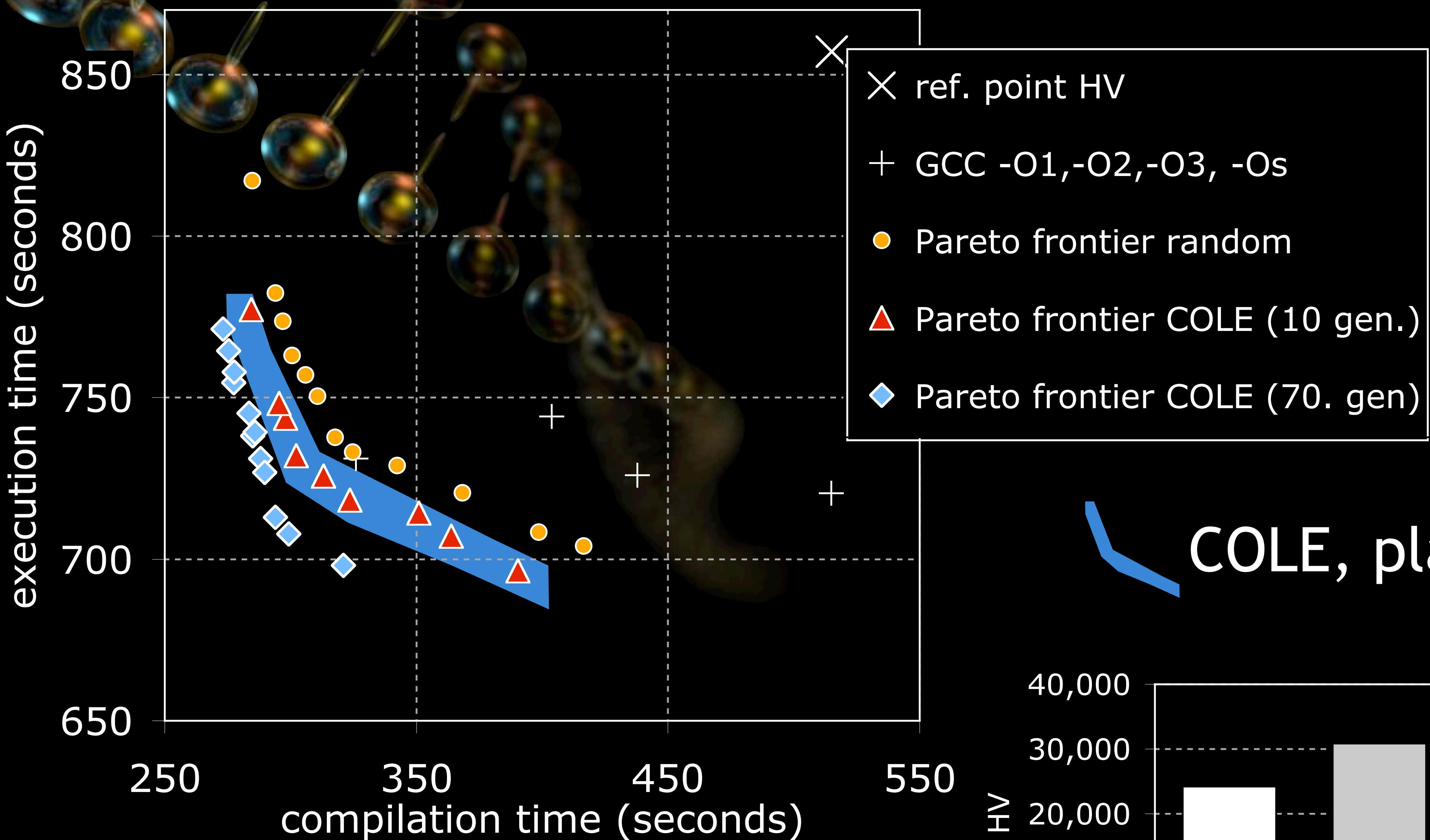


Feeling lucky...

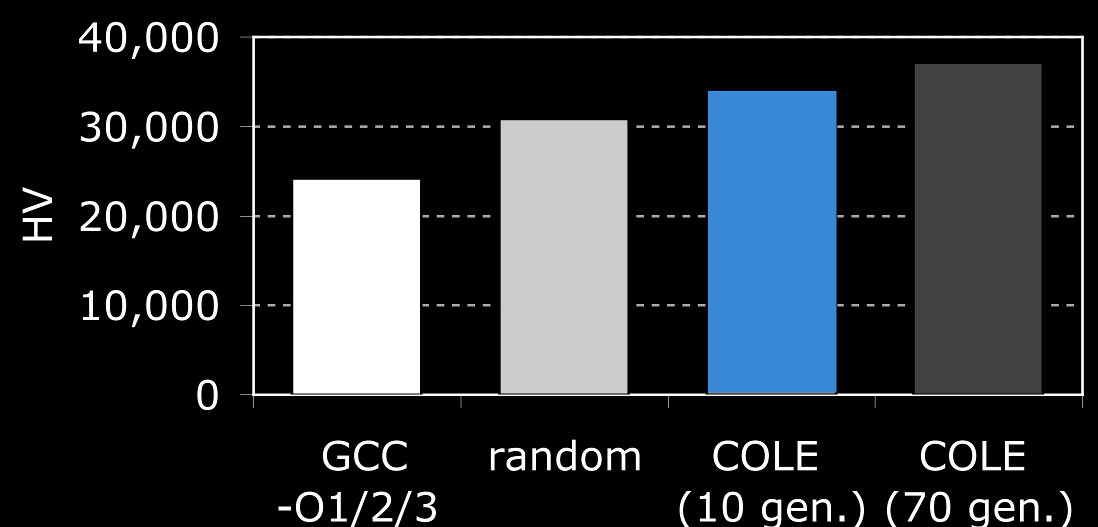


27.4% better in terms of HV than -Ox

And the winner is... COLE!

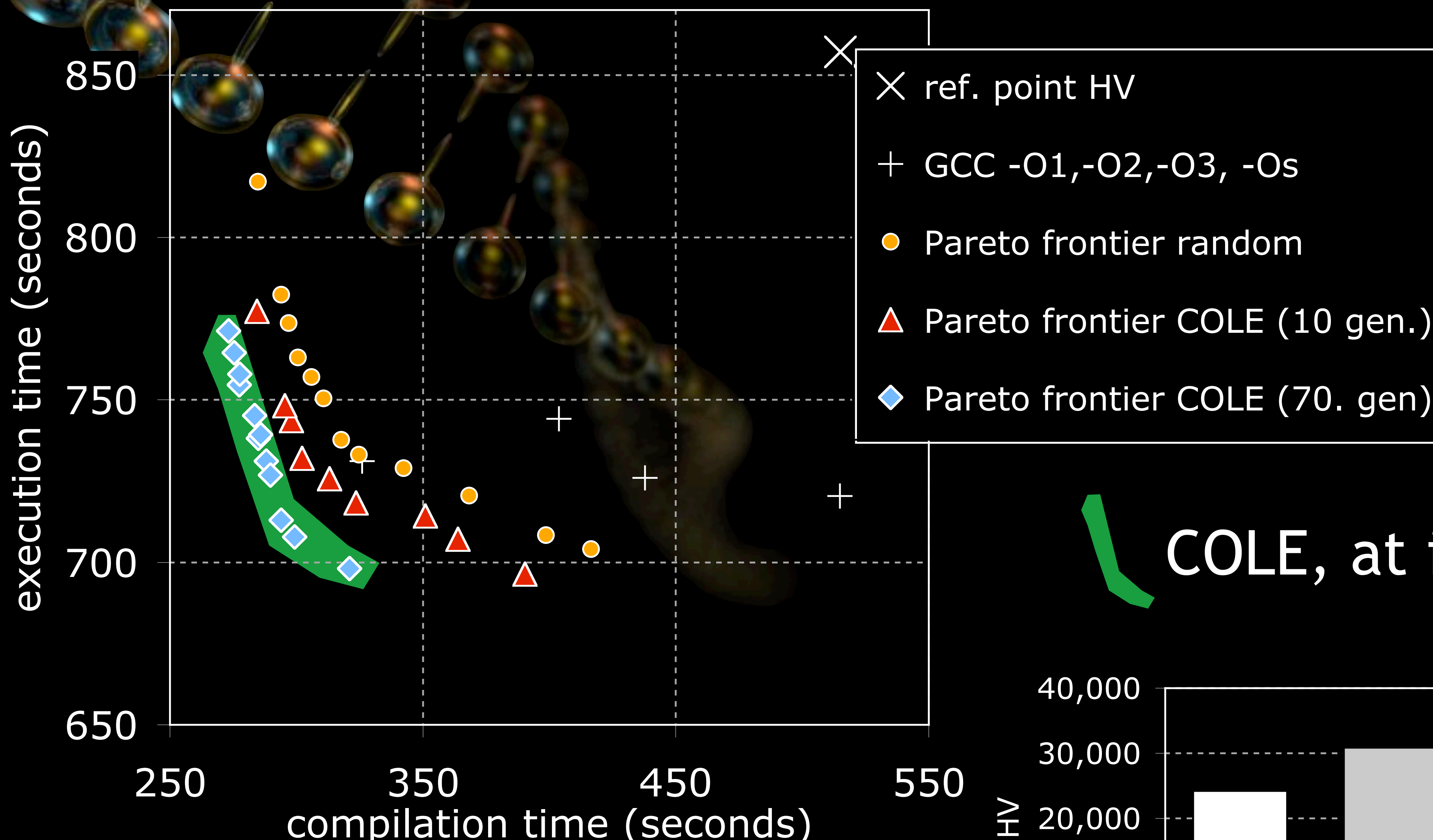


COLE, playing it fair

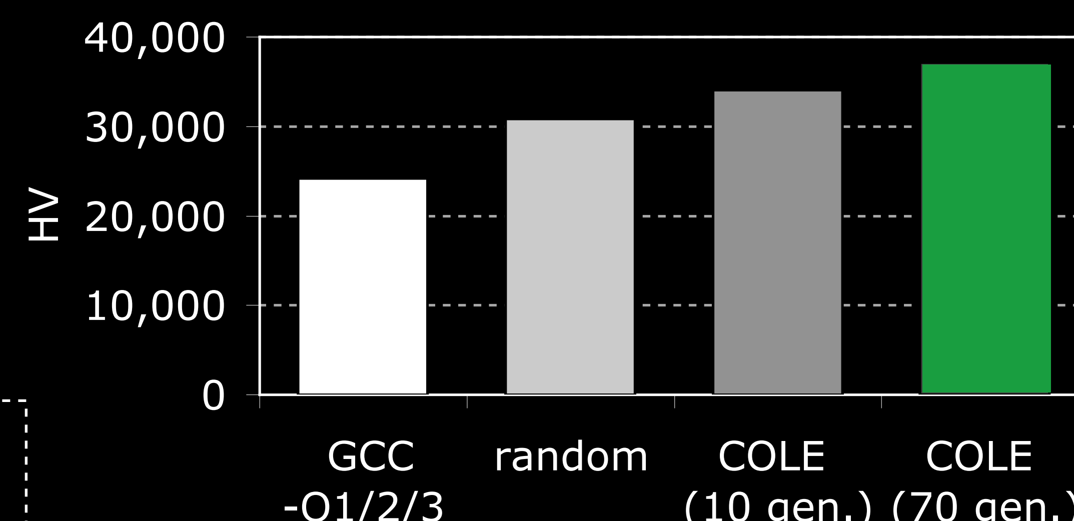


40.6% better than -Ox, 10.4% better than random search with the same search budget

And the winner is... COLE!



COLE, at its best

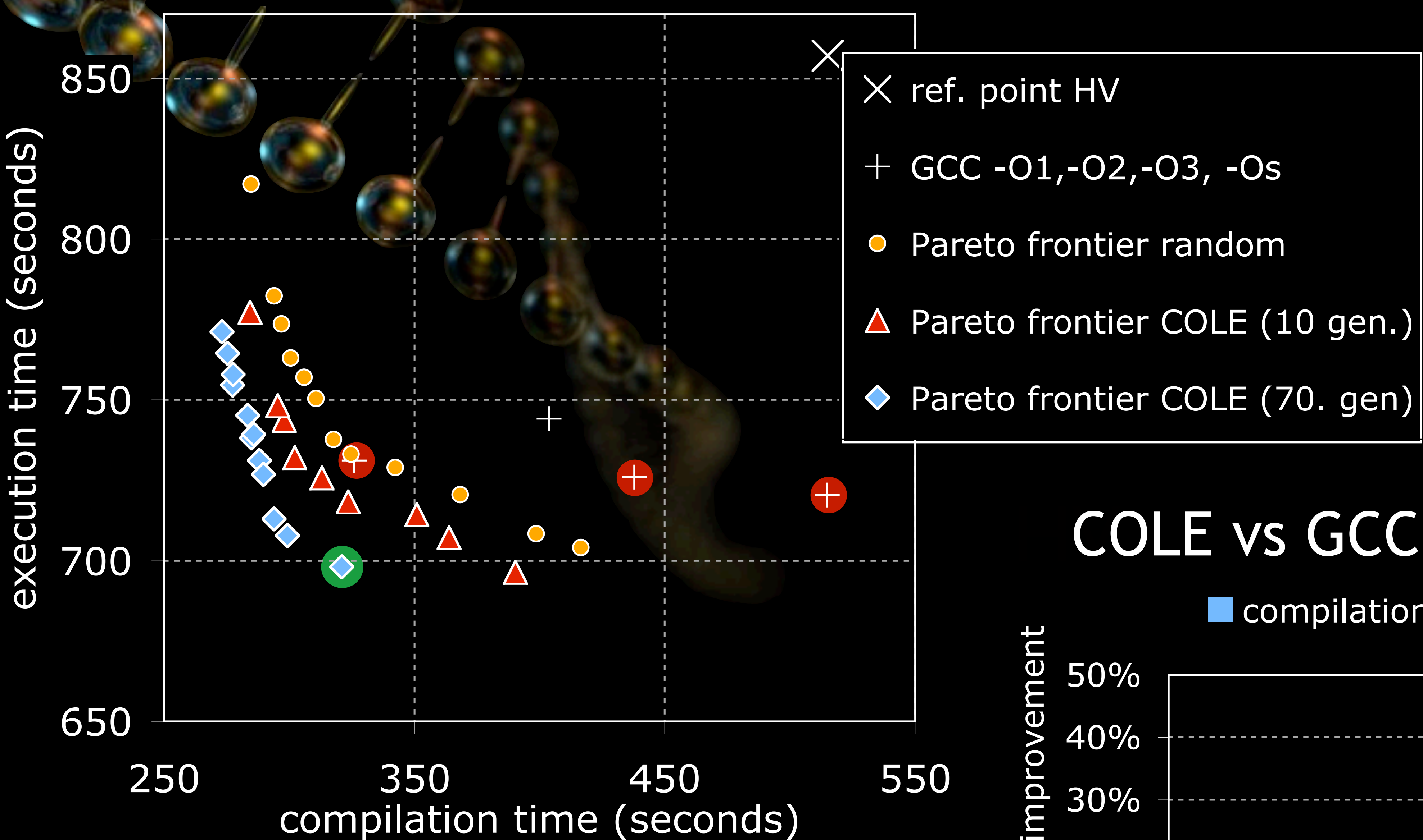


until convergence: 53.4% better than -Ox !

using these optimization levels with

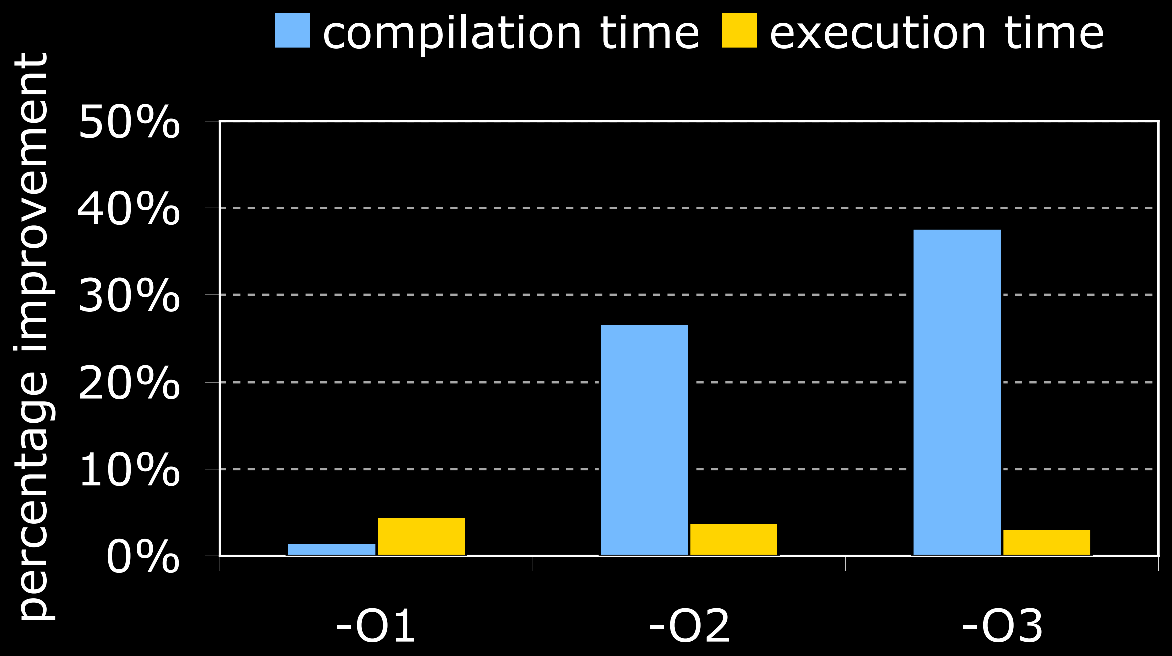
SPEC CPU2006 (train) => 37% better than -Ox

Outperforming the GCC heuristics



- ➡ 4.5% faster than -O1 in 1.5% less comp. time
- ➡ 3.8% faster than -O2 in 26.7% less comp. time
- ➡ 3.1% faster than -O3 in 37.6% less comp. time

COLE vs GCC heuristics



Digging in a little deeper…

optimizations	GCC level	Pareto optimal optimization level											
		0	1	2	3	4	5	6	7	8	9	10	11
-fno-keep-static-consts	default			X									
-fdefer-pop	-O1	X	X	X	X								
-fguess-branch-probability	-O1						X		X		X	X	X
-floop-optimize	-O1											X	X
-ftree-ccp	-O1					X	X	X	X	X	X	X	X
-ftree-dce	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-fre	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-lrs	-O1	X	X	X	X	X	X	X	X	X			
-ftree-sra	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-ter	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-funit-at-a-time	-O1			X	X	X	X	X	X		X	X	X
-fstrength-reduce	-O2		X		X								
-fstrict-aliasing	-O2		X	X	X					X	X	X	X
-ftree-store-copy-prop	-O2	X						X	X	X	X	X	X
-finline-functions	-O3												X
-Os-stripped		X	X	X	X								
-O1-stripped						X	X	X	X	X	X	X	X
-O2-stripped													

Digging in a little deeper…

optimizations	GCC level	Pareto optimal optimization level											
		0	1	2	3	4	5	6	7	8	9	10	11
-fno-keep-static-consts	default			X									
-fdefer-pop	-O1	X	X	X	X								
-fguess-branch-probability	-O1						X		X		X	X	X
-floop-optimize	-O1											X	X
-ftree-ccp	-O1					X	X	X	X	X	X	X	X
-ftree-dce	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-fre	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-lrs	-O1	X	X	X	X	X	X	X	X	X			
-ftree-sra	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-ter	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-funit-at-a-time	-O1			X	X	X	X	X	X		X	X	X
-fstrength-reduce	-O2		X		X								
-fstrict-aliasing	-O2		X	X	X					X	X	X	X
-ftree-store-copy-prop	-O2	X						X	X	X	X	X	X
-finline-functions	-O3												X
-Os-stripped		X	X	X	X								
-O1-stripped						X	X	X	X	X	X	X	X
-O2-stripped													

only 15 of the 60 optimizations are used in the twelve Pareto-optimal optimization levels

Digging in a little deeper...

optimizations	GCC level	Pareto optimal optimization level											
		0	1	2	3	4	5	6	7	8	9	10	11
-fno-keep-static-consts	default			X									
-fdefer-pop	-O1	X	X	X	X								
-fguess-branch-probability	-O1						X		X		X	X	X
-floop-optimize	-O1											X	X
-ftree-ccp	-O1					X	X	X	X	X	X	X	X
-ftree-dce	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-fre	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-lrs	-O1	X	X	X	X	X	X	X	X	X			
-ftree-sra	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-ter	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-funit-at-a-time	-O1			X	X	X	X	X	X		X	X	X
-fstrength-reduce	-O2		X		X								
-fstrict-aliasing	-O2		X	X	X					X	X	X	X
-ftree-store-copy-prop	-O2	X						X	X	X	X	X	X
-finline-functions	-O3												X
-Os-stripped		X	X	X	X								
-O1-stripped						X	X	X	X	X	X	X	X
-O2-stripped													

SSA tree optimizations are quite effective

6/15, four appear in all 12 Pareto-optimal opt. levels

Digging in a little deeper…

optimizations	GCC level	Pareto optimal optimization level											
		0	1	2	3	4	5	6	7	8	9	10	11
-fno-keep-static-consts	default			X									
-fdefer-pop	-O1	X	X	X	X								
-fguess-branch-probability	-O1						X		X		X	X	X
-floop-optimize	-O1											X	X
-ftree-ccp	-O1					X	X	X	X	X	X	X	X
-ftree-dce	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-fre	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-lrs	-O1	X	X	X	X	X	X	X	X	X			
-ftree-sra	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-ter	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-funit-at-a-time	-O1			X	X	X	X	X	X		X	X	X
-fstrength-reduce	-O2		X		X								
-fstrict-aliasing	-O2		X	X	X					X	X	X	X
-ftree-store-copy-prop	-O2	X						X	X	X	X	X	X
-finline-functions	-O3												X
-Os-stripped		X	X	X	X								
-O1-stripped						X	X	X	X	X	X	X	X
-O2-stripped													

some optimizations are expensive, but benefit code quality

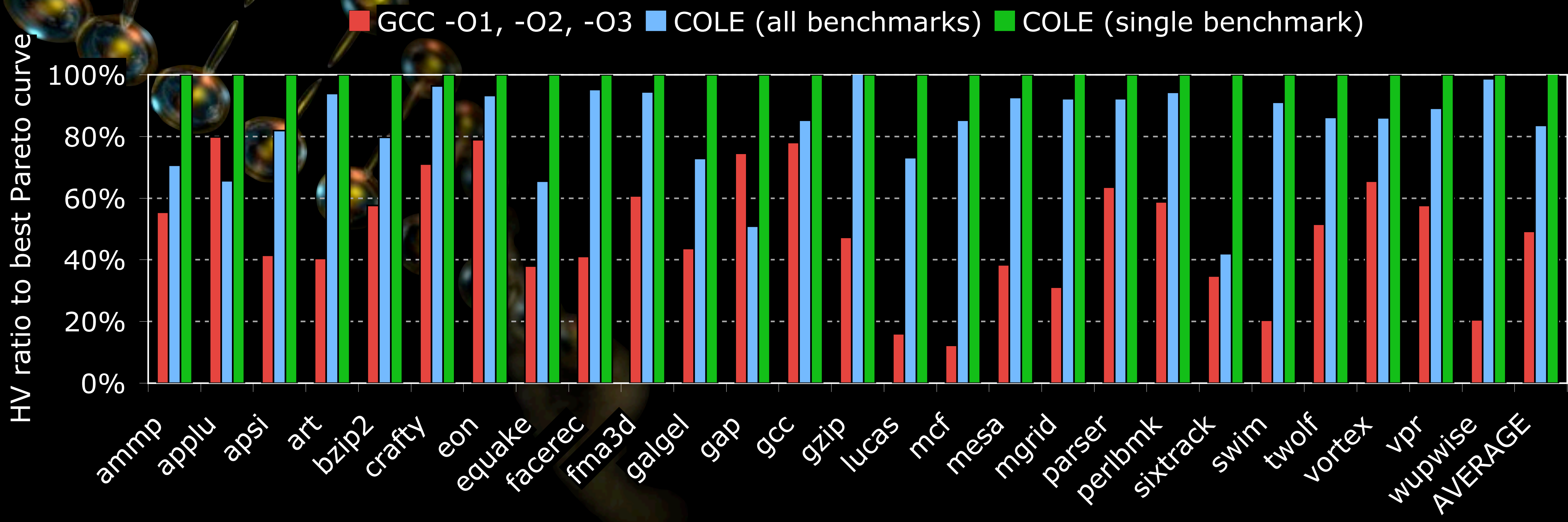
Digging in a little deeper...

optimizations	GCC level	Pareto optimal optimization level											
		0	1	2	3	4	5	6	7	8	9	10	11
-fno-keep-static-consts	default			X									
-fdefer-pop	-O1	X	X	X	X								
-fguess-branch-probability	-O1						X		X		X	X	X
-floop-optimize	-O1											X	X
-ftree-ccp	-O1					X	X	X	X	X	X	X	X
-ftree-dce	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-fre	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-lrs	-O1	X	X	X	X	X	X	X	X	X			
-ftree-sra	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-ftree-ter	-O1	X	X	X	X	X	X	X	X	X	X	X	X
-funit-at-a-time	-O1			X	X	X	X	X	X		X	X	X
-fstrength-reduce	-O2		X		X								
-fstrict-aliasing	-O2		X	X	X					X	X	X	X
-ftree-store-copy-prop	-O2	X						X	X	X	X	X	X
-finline-functions	-O3												X
-Os-stripped		X	X	X	X								
-O1-stripped						X	X	X	X	X	X	X	X
-O2-stripped													

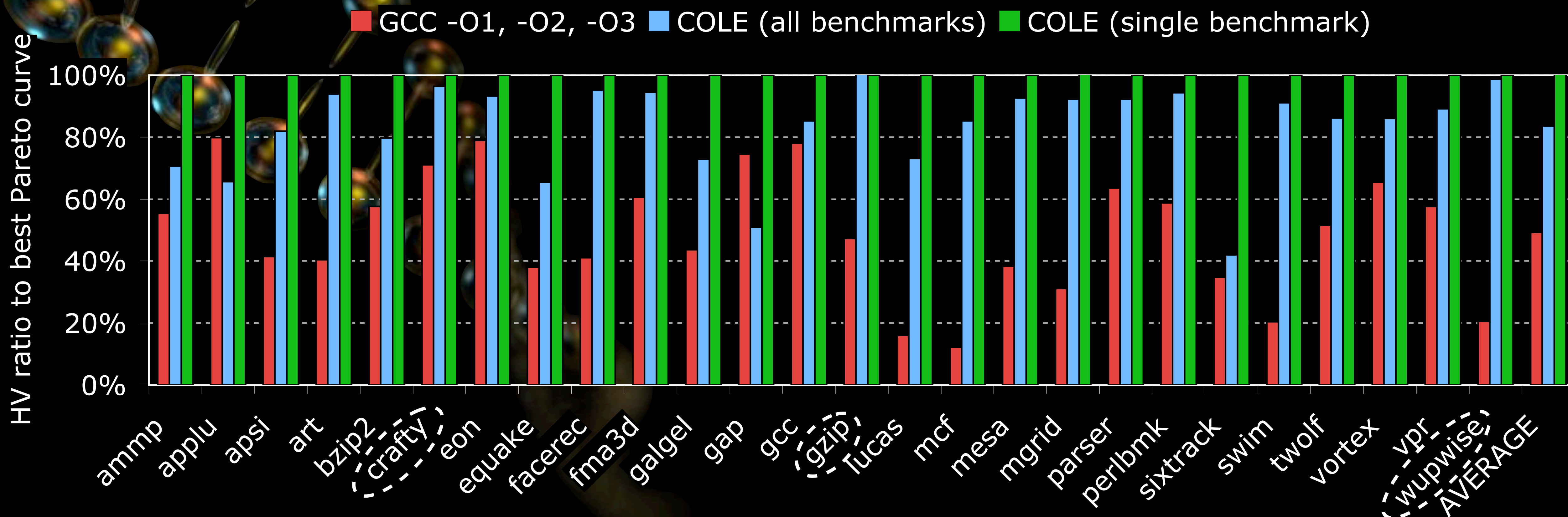
-Os is the best base for faster compilation

-O1 is the best base for better code quality

Sensitive or not?



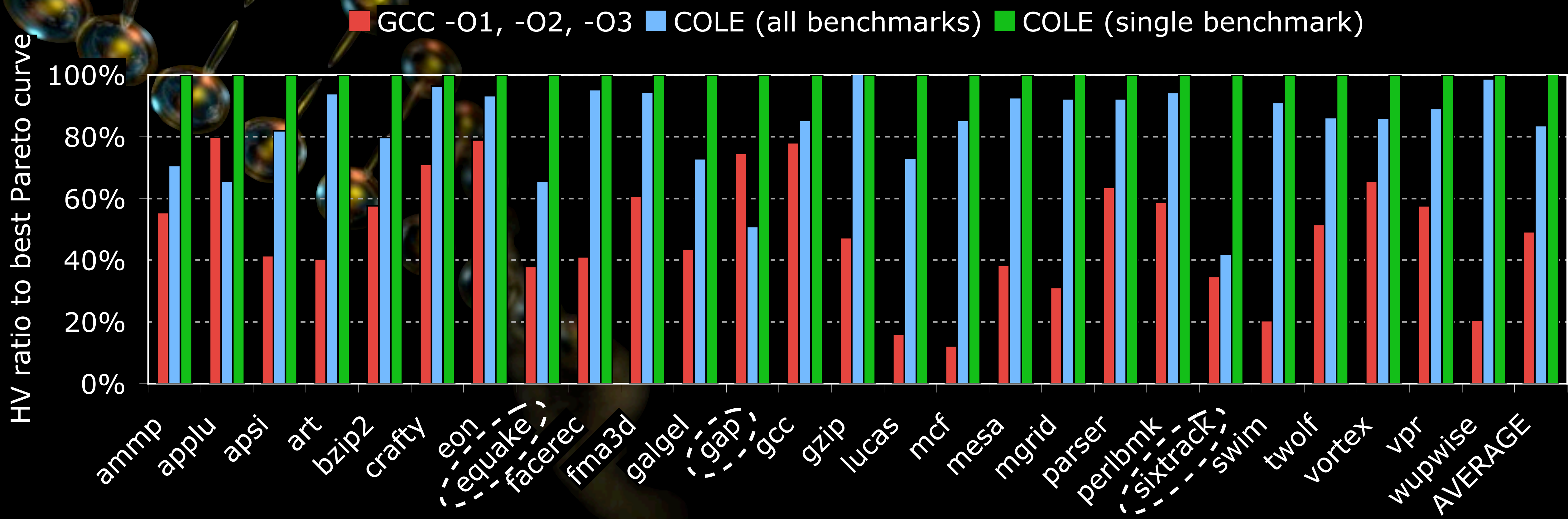
Sensitive or not?



➡ some benchmarks follow the overall trend in optimization levels...

crafty, gzip, wupwise, ...

Sensitive or not?



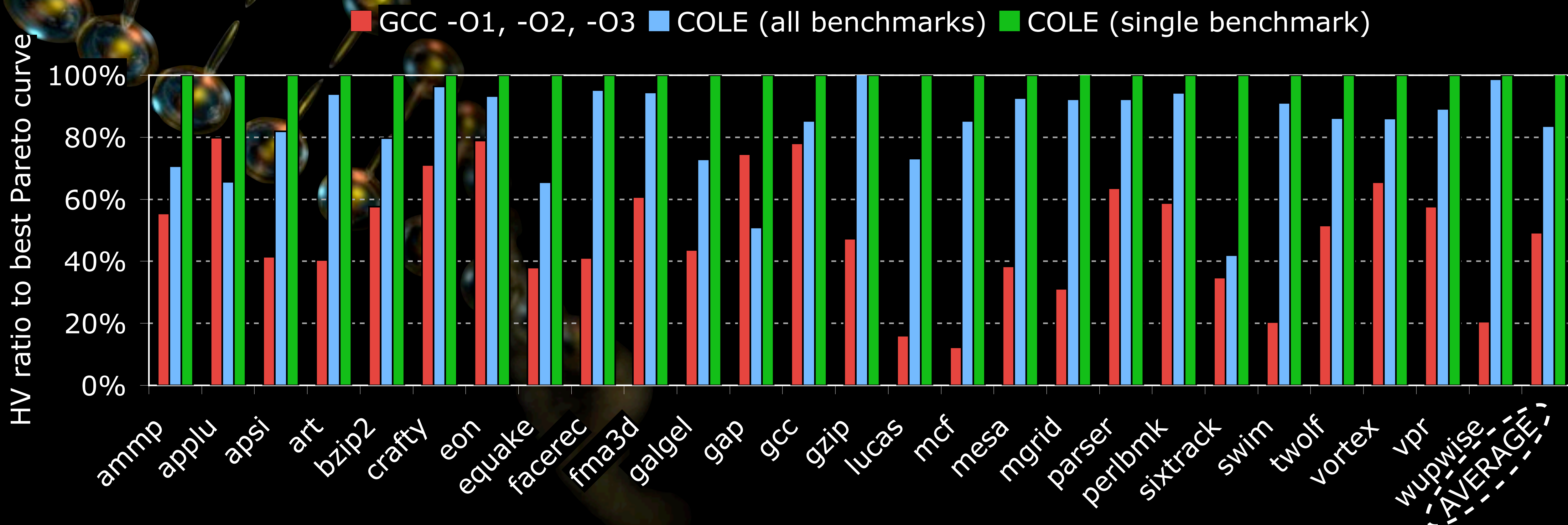
➡ some benchmarks follow the overall trend in optimization levels...

crafty, gzip, wupwise, ...

➡ ... others show very specific Pareto-optimal optimization levels

equake, gap, sixtrack, ...

Sensitive or not?



- ➡ some benchmarks follow the overall trend in optimization levels...

  crafty, gzip, wupwise, ...

- ➡ ... others show very specific Pareto-optimal optimization levels

  equake, gap, sixtrack, ...

- ➡ overall: COLE gets fairly close to the per-benchmark optimal trade-off

We want more!

We believe this work will trigger further research:

➡ room for improvement in terms of speed

pruning the search space
reducing time spent evaluating entities
benchmark subsetting

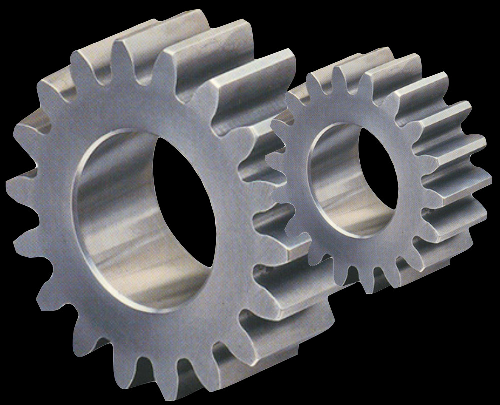
➡ study ordering of optimization passes (using GCC-ICI)

➡ identify important optimizations for various platforms

➡ also take non-Ox optimizations into consideration to focus development on most interesting optimizations

➡ see how a compiler suite evolves over time in terms of suitable optimizations

Conclusions



- COLE allows for automated construction of candidate optimization levels
- clearly outperforms optimization levels based on heuristics and random search
- analyzing optimizations used in Pareto-optimal points yields various interesting insights
- identifying optimization-sensitive benchmarks may be helpful for compiler developers

COLE: Compiler Optimization Level Exploration

Kenneth Hoste and Lieven Eeckhout
Ghent University, Belgium

CGO-2008, Boston (MA)
April 8th, 2008

UNIVERSITEIT GENT