Concurrent Root Cut Loops to Exploit Performance Variability

Andrea Tramontani
CPLEX Optimization, IBM Italy

Joint work with

Outline

- Performance variability
- Exploiting performance variability for performance improvement
  - Distributed concurrent MIP in CPLEX 12.6.0
- Concurrent root cut loops
  - External Implementation with CPLEX 12.5.0
  - Internal Implementation and Performance Impact in CPLEX 12.5.1

Performance Variability

- Performance variability is a well known **issue intrinsic to MIP**:
  - Danna (MIP Workshop, 2008)
  - Koch et al. (Math. Prog. Comp., 2011)
  - Fischetti and Monaci (Op. Res., to appear)
  - Achterberg and Wunderling (Facets of Comb. Optimization, 2013)
  - Lodi and T. (INFORMS 2013 Tutorial)

- MIP Solvers contain various ingredients:
  - Heuristics
  - Cutting planes
  - Criteria for branching variable selection
  - ...

- Seemingly performance neutral changes (in the platform, in the code, or in the parameter setting) may have a big impact
  - on all those ingredients (separated cuts, heuristic solution found, picked variable to branch on)
  - and therefore on the whole solution process

Performance Variability (Cont.d)

- A natural source of performance variability is the **choice of the initial LP basis**
- Equivalent (but different) optimal solution/bases of the first LP relaxation may lead to different
  - cutting planes
  - heuristic solutions
  - ...
- Reoptimizing with different cutting planes amplifies the diversification
- The final root node could be very different in terms of
  - Fractional solution
  - Dual and primal bounds
  - Cuts active in the LP
- Branching rules will take different decisions

Simulate Performance Variability: an experiment with the random seed

Compare:

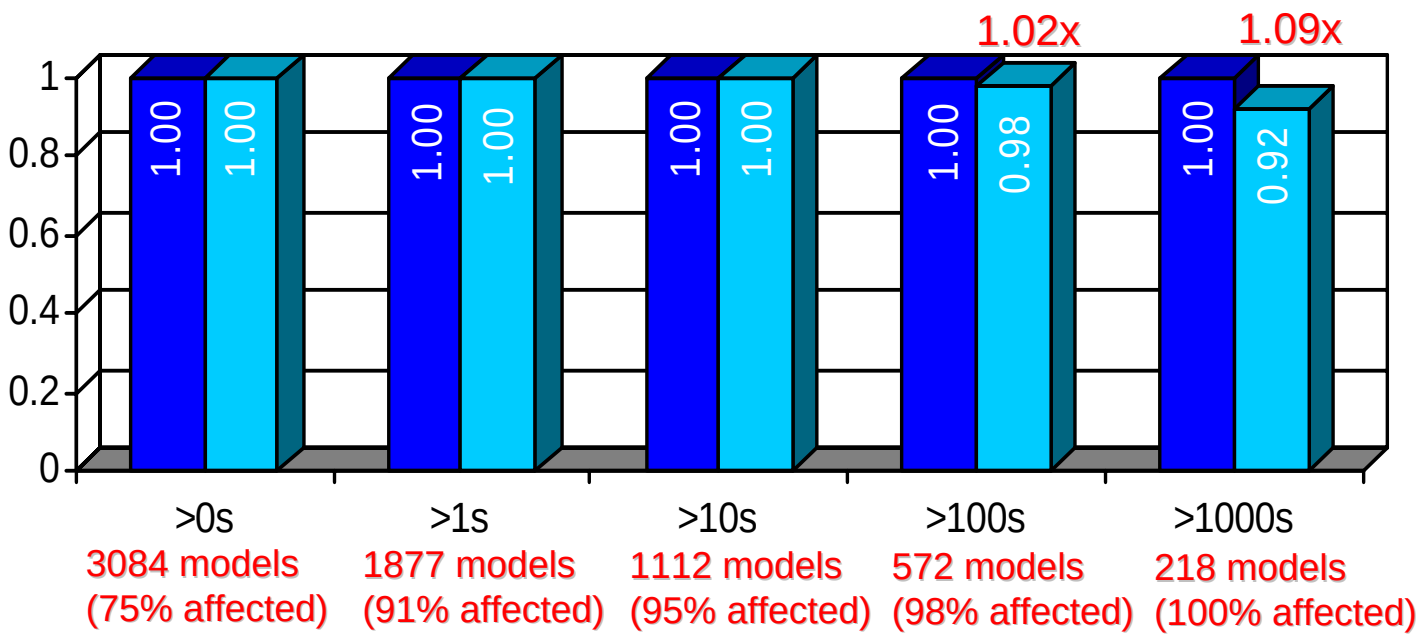
- Solver A: CPLEX 12.6.0 with **random seed 1**
- Solver B: CPLEX 12.6.0 with **random seed 2**

- Test set: 3224 models classified according to their difficulty:
  - [0, 10k]: all models but the ones for which the solvers provide an inconsistent answer
  - [*T*, 10k]: all models for which one of the solver takes at least *T* seconds

- The **comparison is fair** because the difficulty of a given model is defined by all the compared solver: there is **no bias** against any of the compared solvers

- Comparison: geomean of computing times

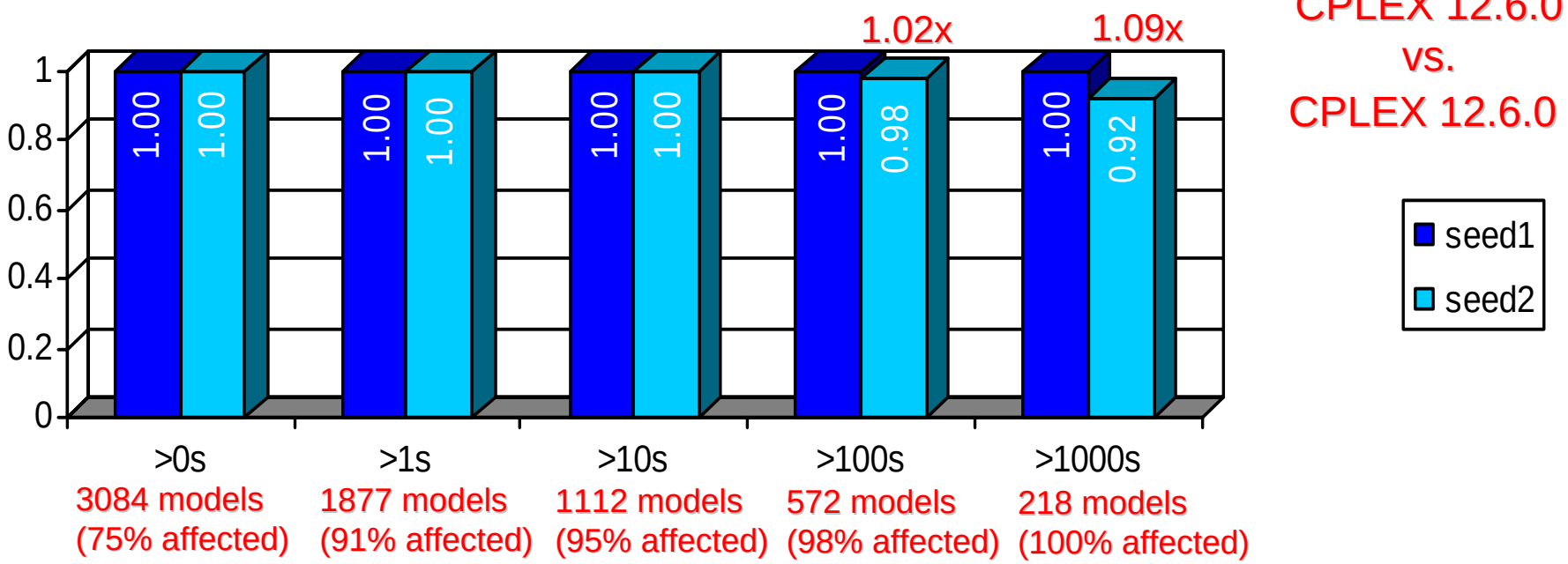
Performance Variability: does it matter?



CPLEX 12.6.0
vs.
CPLEX 12.6.0

Date: 20 December 2013
Testset: 3224 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

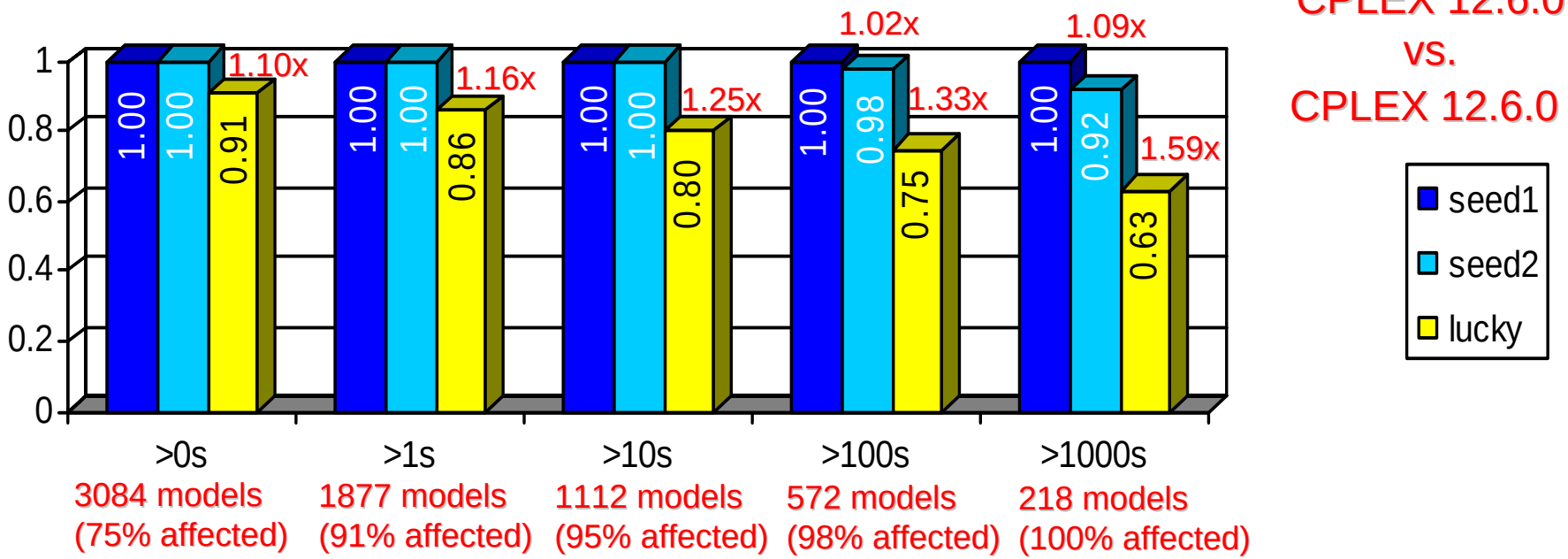
Performance Variability: does it matter?



- 91% of the models in [1,10k] affected:
  the random seed is effective to simulate performance variability.
- More than 200 models in [1k,10k] can not measure performance difference of less than 10%:
  - Small test sets are less robust to outliers,
  - Harder models are typically the ones exhibiting the larger variability.

Date: 20 December 2013
Testset: 3224 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

Performance Variability: good news from the lucky solver



- **Lucky solver**: for each model, take the best time among the competitors.

Date: 20 December 2013
Testset: 3224 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

Exploiting Performance Variability for Performance Improvement

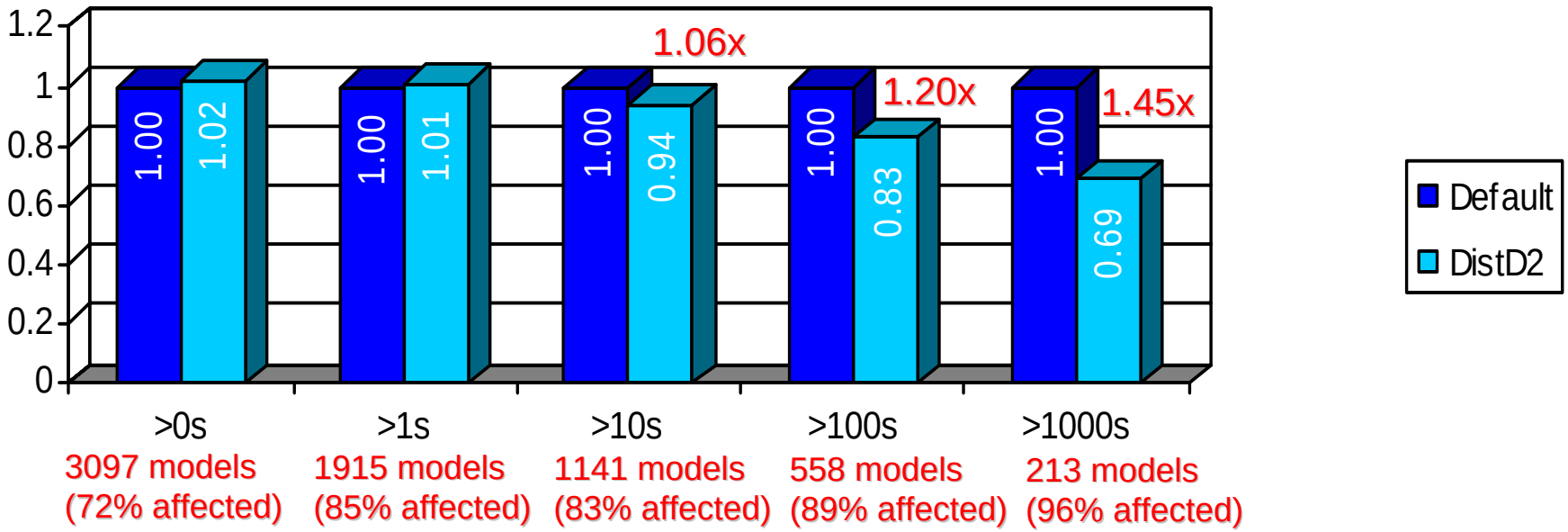
- Fischetti and Monaci (Op. Res., to appear):
  - Run CPLEX k times, each time rooted at a different initial LP basis, for few nodes
  - Bet the winner and let it run up to completion
- Carvajal, Ahmed, Nemhauser, Furman, Goel, Shao (Opt. Online, 2013):
  - Run k single threaded branch-and-cuts with different parameter setting instead of one single branch-and-cut with k threads
  - Different strategies to share information are tested
- Fischetti, Lodi, Monaci, Salvagnin, T. (submitted):
  - Concurrent root cut loops

Distributed Concurrent and Distributed Parallel MIP in CPLEX 12.6.0

- **Deterministic Distributed Parallel MIP**
  - Similar to **Yuji Shinano et al.** (ParaSCIP and ParaCPLEX), but deterministic
  - One master process coordinates k workers on a deterministic basis
  - Racing **ramp-up** phase
    - all workers solve model, each with different parameter configuration
    - regular synchronization based on deterministic time (report status and share primal bounds)
    - automatic stop criterion selects winner
  - **Distributed parallel branch-and-cut** phase
    - Distribute some nodes from ramp-up winner to other workers
    - Run the winner up to completion

- **Deterministic Distributed Concurrent MIP**
  - Just do **infinite horizon ramp-up**
  - This is the option active by default

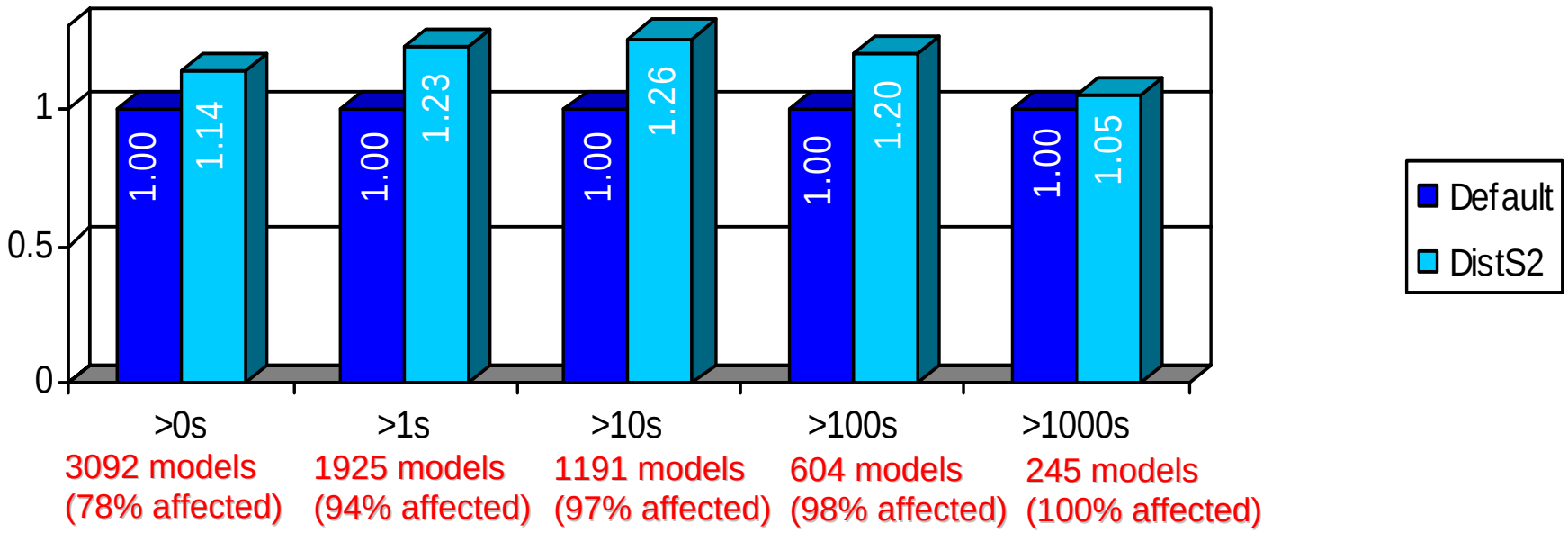
Infinite horizon ramp-up on different machines



- **Default:** CPLEX 12.6.0 on one 12 cores machine
- **DistD2:** Infinite horizon ramp-up with **two workers on two 12 cores machines**

Date: 20 December 2013
Testset: 3224 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

Infinite horizon ramp-up on the same machine



- **Default:** CPLEX 12.6.0 on one 12 cores machine
- **DistS2:** Infinite horizon ramp-up with **two workers on the same 12 cores machine**:
  - Each worker is a 6 threads run.

Date: 20 December 2013
Testset: 3224 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

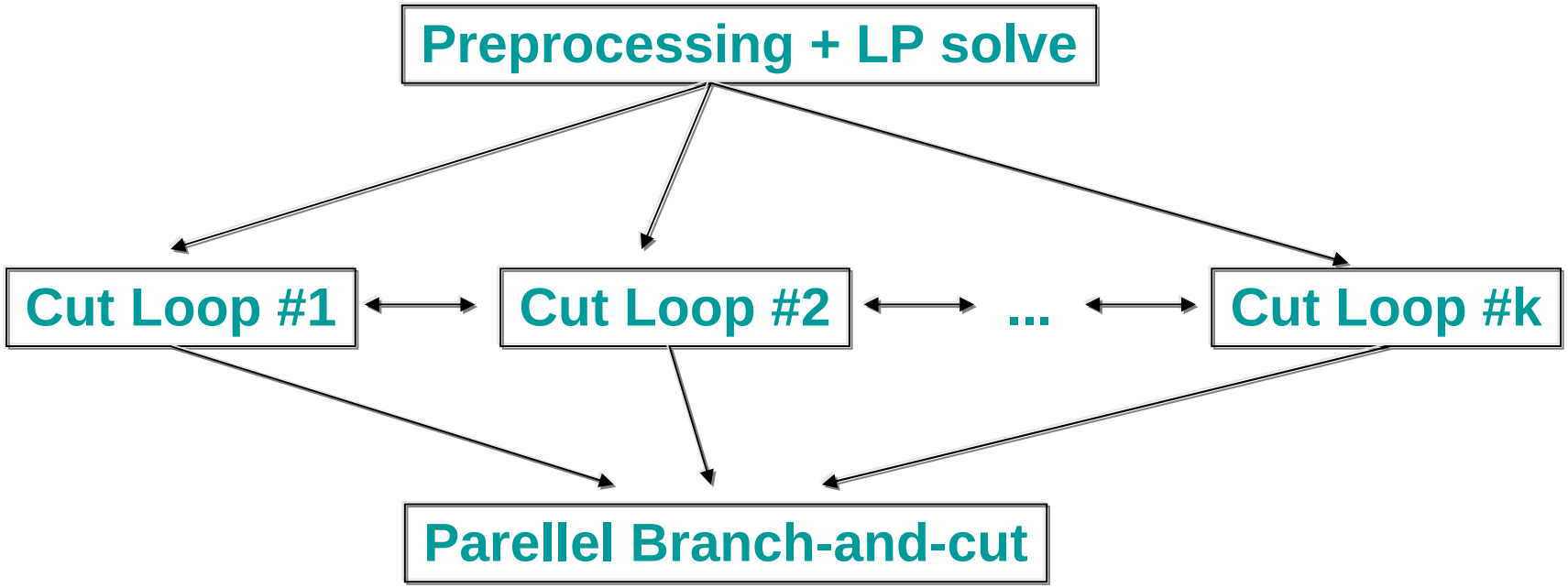
Infinite horizon ramp-up: Lesson learned

- Running k different **branch-and-cut** (each one with different parameter configuration) is
  - Good in a distributed memory environment
  - Bad in a shared memory environment

- Idea (already exploited in CPLEX 12.5.1):
  Exploit performance variability **only at the root node**, to
  - Collect more cutting planes
  - Find a (possibly) better incumbent solution

Concurrent Root Cut Loops

- More classical context: one single machine with multi cores (shared memory)
- Exploit multi-threading and performance variability at **root node**:
  - Run k concurrent root cut loops in a parallel fashion, each one rooted at a different LP basis
  - Along the process, share cuts and feasible solutions among the k cut loops
- Then do regular parallel branch-and-cut



External implementation: CPLEX 12.5.0 with callbacks

- Preprocess all the instances once, then run all the code with 1 thread only and presolver disabled.
- Sampling phase:

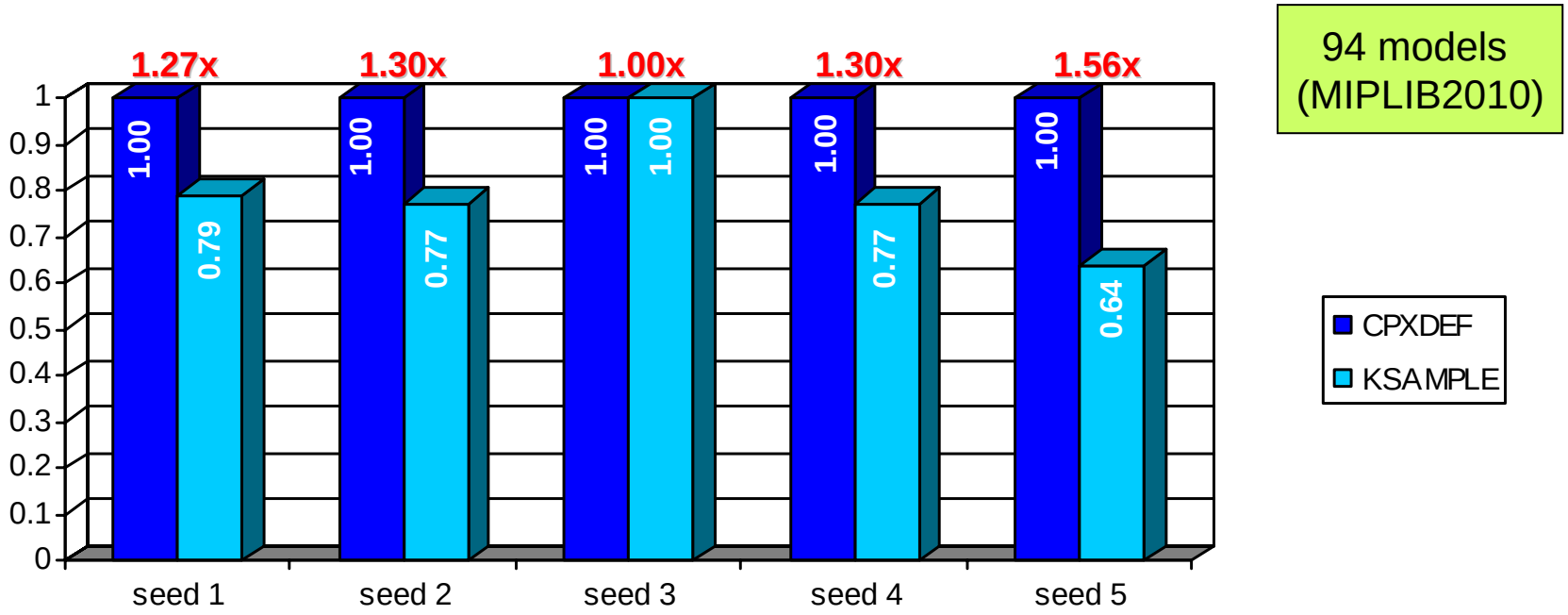
  For h = 1, ..., k-1
  - Solve the root node using **random seed** h
  - store **cuts and incumbent** solution available at the end of the root node in a pool
- Final run using random seed k: compare
  - CPXDEF: CPLEX with an empty usercut callback installed
  - KSAMPLE: CPLEX with
    - Usercut callback that separate cuts from the pool only once at the end of root node
    - Heuristic callback that installs the incumbent only once at the end or root node.

External Implementation: test set and fake assumption

- Sampling phase is done with k = 10
- Final run (CPXDEF vs. KSAMPLE) is done with 5 different random seeds
- Test set made of publicy available MIP models: MIPLIB2010 benchmark and primal test sets
- Fake assumption:
  - We assume multi-threading is for free and we do not include the time for the sampling phase in KSAMPLE
  - 23 models solved in the sampling phase are removed from the test set:

    We are left with 94 models

External Implementation: computational results

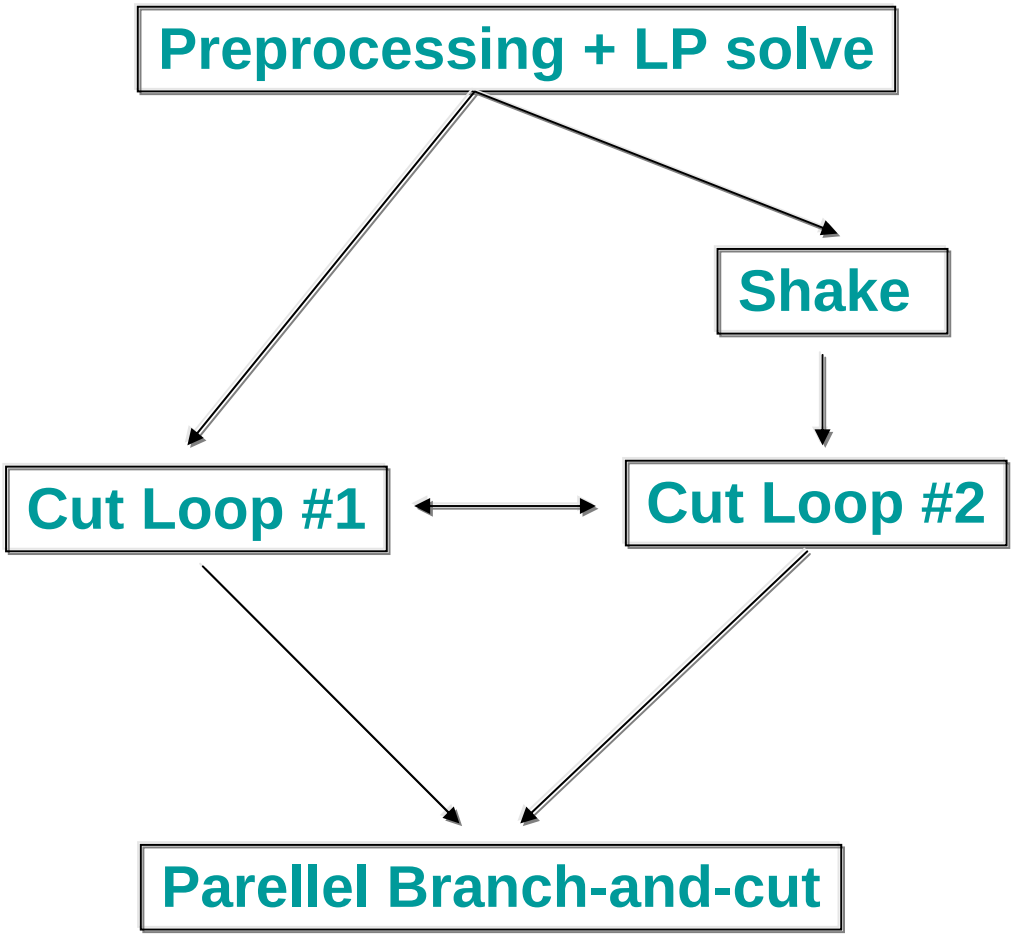


- The results indicate the idea has some potential, but
  - Disregarding the time for the sampling phase is a **big bias** against CPXDEF
  - CPXDEF is far away from CPLEX default
  - **94 models only** can just give some insights (speed up varies **from 1.56x to 1.00x** depending on the random seed)

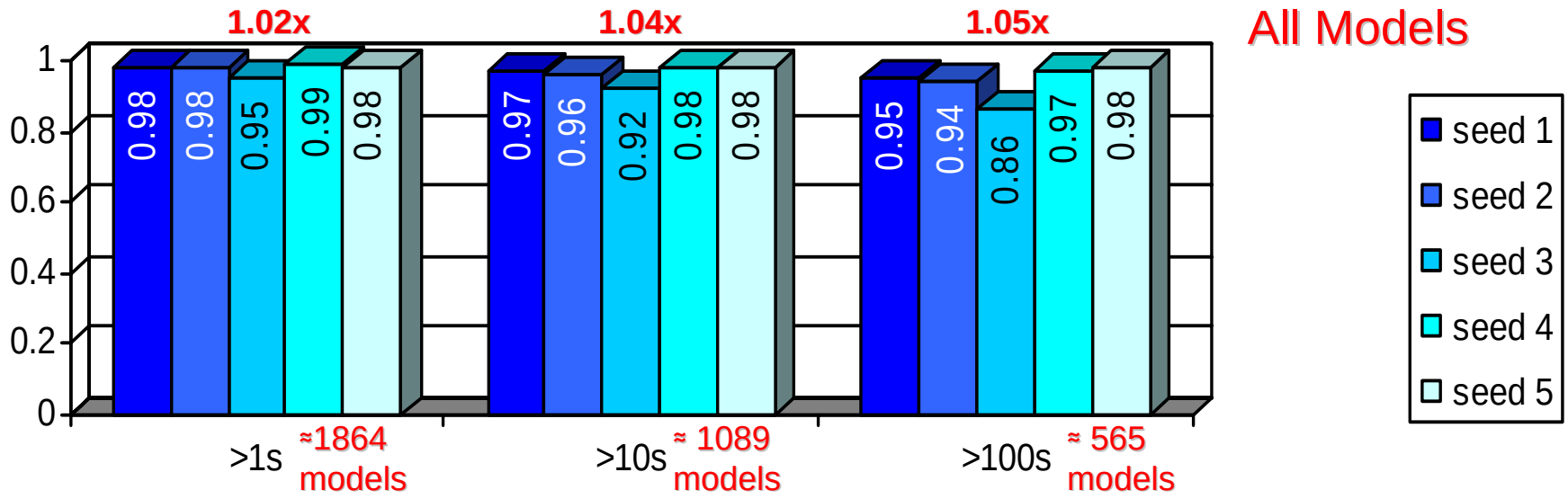
Internal Implementation in CPLEX 12.5.1

- Forget about large K (we just made it with K = 2)
  - Multi-threading is not for free
  - We also want to apply other methods at the root node
  - More cuts typically means better dual bound,
    better dual bound does not mean less time to solve
- Create diversification in the parallel cut loop by changing
  - The basis
  - The random seed
  - Other parameters
- Be conservative with the cuts:
  - share cuts between the two cut loops,
  - but be clever when selecting the cuts to be shared

Internal Implementation in CPLEX 12.5.1 (Cont.d)

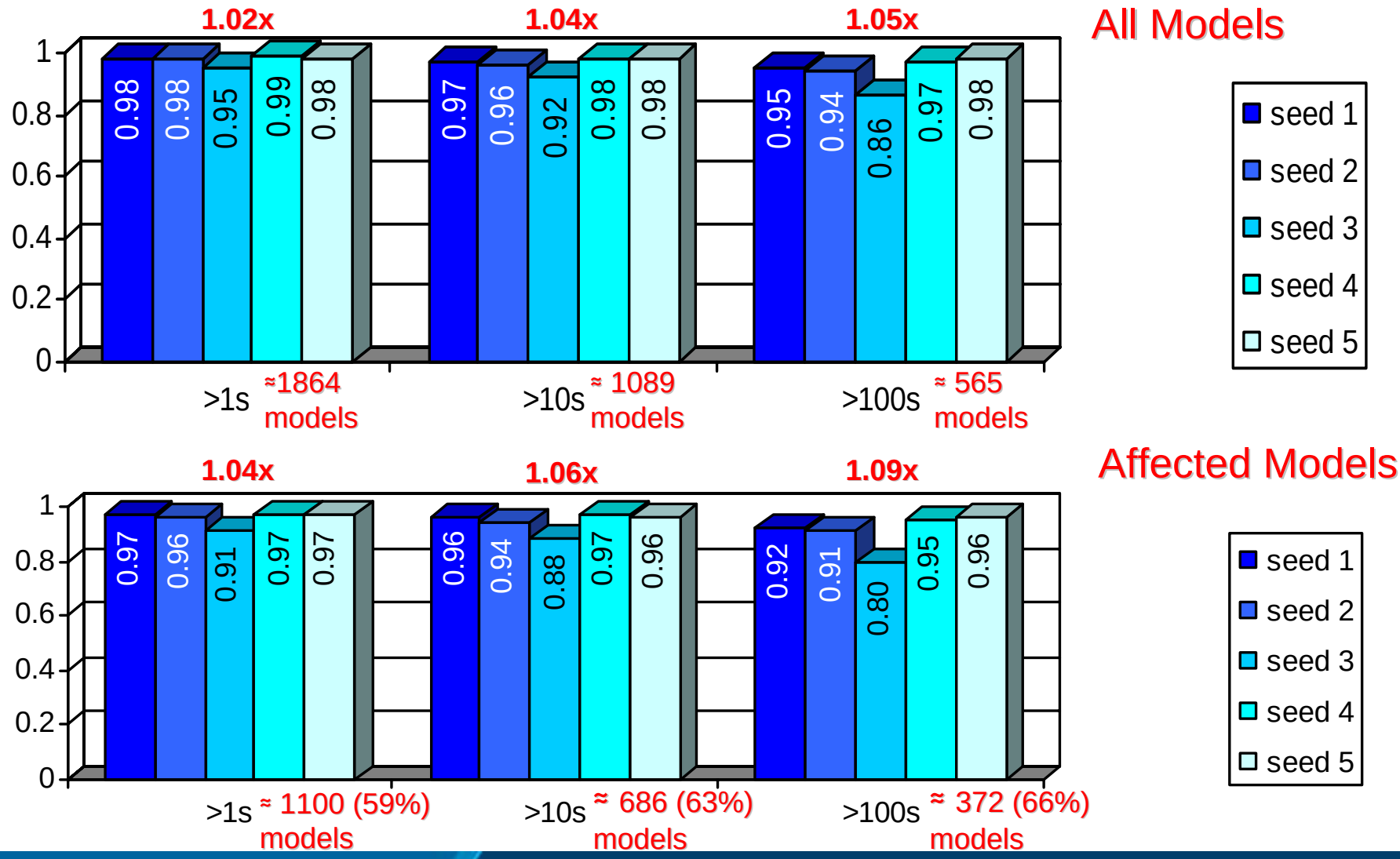


Concurrent Root Cut Loops: Performance Impact in CPLEX 12.5.1



Date: 23 May 2013
Testset: 3223 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

Concurrent Root Cut Loops: Performance Impact in CPLEX 12.5.1



Date: 23 May 2013
Testset: 3223 models
Machine: Intel X5650 @ 2.67GHz, 24 GB RAM, 12 threads
Timelimit: 10,000 sec

More work to be done ...

- More cut loops means **more information**:
  - More cutting planes (that must be filtered aggressively)
  - More fractional points to be used for the filtering
- With the current cut selection strategy,
  - K = 3 cut loops is roughly a 5% performance degradation w.r.t. K = 2
- Moving to K > 2 cut loops asks for more effective cut selection strategies