Parallelism in LP and MIP

Thank you for joining us. We will be starting shortly.

GUROBI OPTIMIZATION

The World's Fastest Solver

Today's Speaker

Ed Rothberg

CEO, Co-Founder

Gurobi Optimization

How to Exploit Parallelism in Linear Programming and Mixed-Integer Programming

Ed Rothberg, CEO & Co-founder, Gurobi Optimization

GUROBI OPTIMIZATION

The World's Fastest Solver

Outline

- **Parallelism in LP**
- **Parallelism in MIP**
- **Distributed algorithms**
  - Distributed MIP
  - Distributed tuning
  - Distributed concurrent
- **Other metrics**
  - Maximizing throughput on a parallel server
- **Other architectures**
  - Graphical Processing Unit (GPU)
  - Quantum computer

Important Concepts

Sources of Parallelism

- **We exploit 3 sources of parallelism in Gurobi**
  - Parallel algorithms
    - Divide up 'fixed' pile of work
  - Diversity of algorithms
    - Given a mix of algorithms…
      - Run them all at once
      - Stop when the first one finishes
  - Performance variability
    - Given an algorithm that can experience large performance swings on the same problem…
      - Run multiple instances at once with multiple settings
      - Stop when the first one finishes

Non-determinism

- **Parallel algorithms often exhibit non-deterministic behavior**
  - Same problem, same machine -> different (equivalent) results
- **Nearly all Gurobi parallel algorithms are deterministic**
  - Same problem, same machine -> same result *every time*
- **A few Gurobi parallel algorithms exhibit mild non-determinism**
  - A small number of possible outcomes
  - Examples:
    - Concurrent LP
    - Concurrent MIP
- **We avoid highly non-deterministic algorithms**

Determinism – How?

- **Basic principle for avoiding non-determinism…**
  - *Which thread finished first?*
  - Answer must be the same *every time*
- **Options:**
  - Wall-clock time: not reliable
  - CPU counters: not exposed, few are reliable
  - Instrument the code
- **Every algorithm in Gurobi makes an estimate of how much work it did**
  - *Which thread finished first?*
  - The one with the smaller work estimate

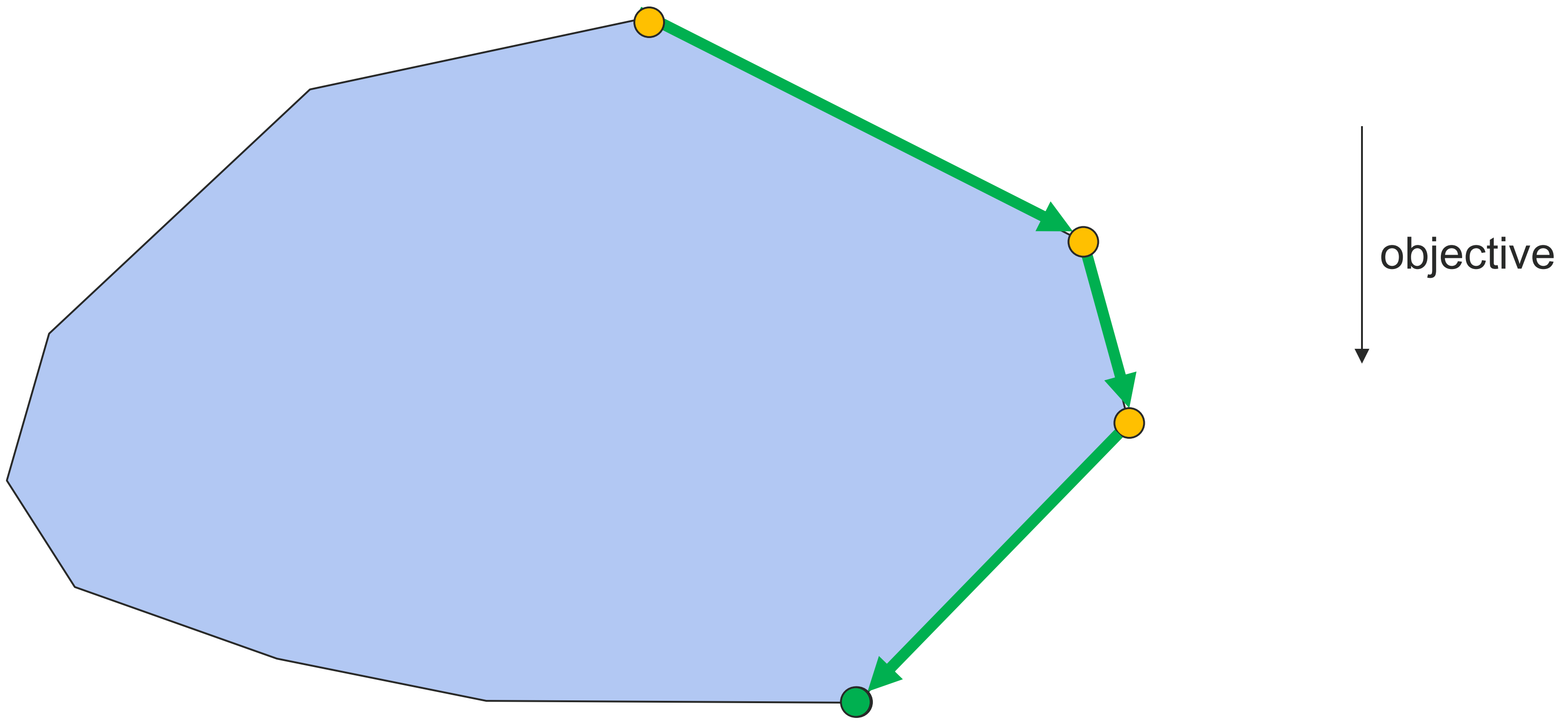
Parallelism in LP

Two Fundamental Algorithms for Linear Programming

- Simplex algorithm (primal and dual)
- Interior point (barrier) algorithm

- When considering parallelism in LP, more like…
  - One workhorse (simplex)
  - Plus one very sophisticated initial basis selection procedure (barrier)

Simplex Algorithm



- Iterate until no more improving direction is found
  - This is an optimal solution to the LP.

Simplex Algorithm – Linear Algebra

- **Primal feasibility constraints**

$$Ax = b$$

- **Partition into basic and non-basic variables:** $A = (B, N)$
  - Non-basic structural variables correspond to tight bounds
  - Non-basic slack variables correspond to tight constraints

$$Bx_B + Nx_N = b$$

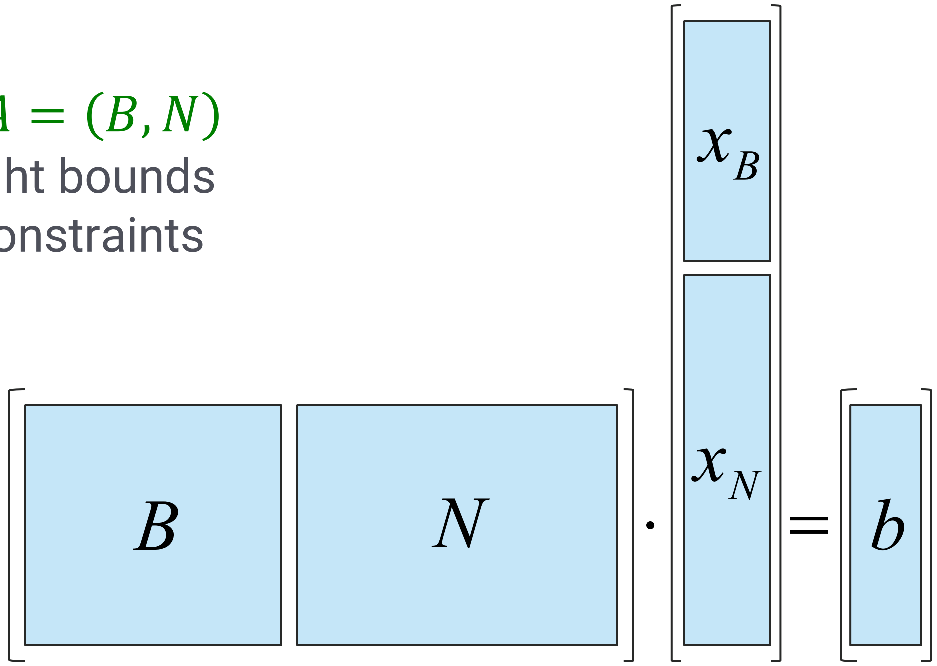
- **Solve for basic variables**

$$x_B = B^{-1}(b - Nx_N)$$

- **Solved by maintaining**

$$B = LU$$

- **Perform a sequence of pivots**
  - Swap one non-basic variable for one basic variable
  - Update basis matrix (and basis factor)

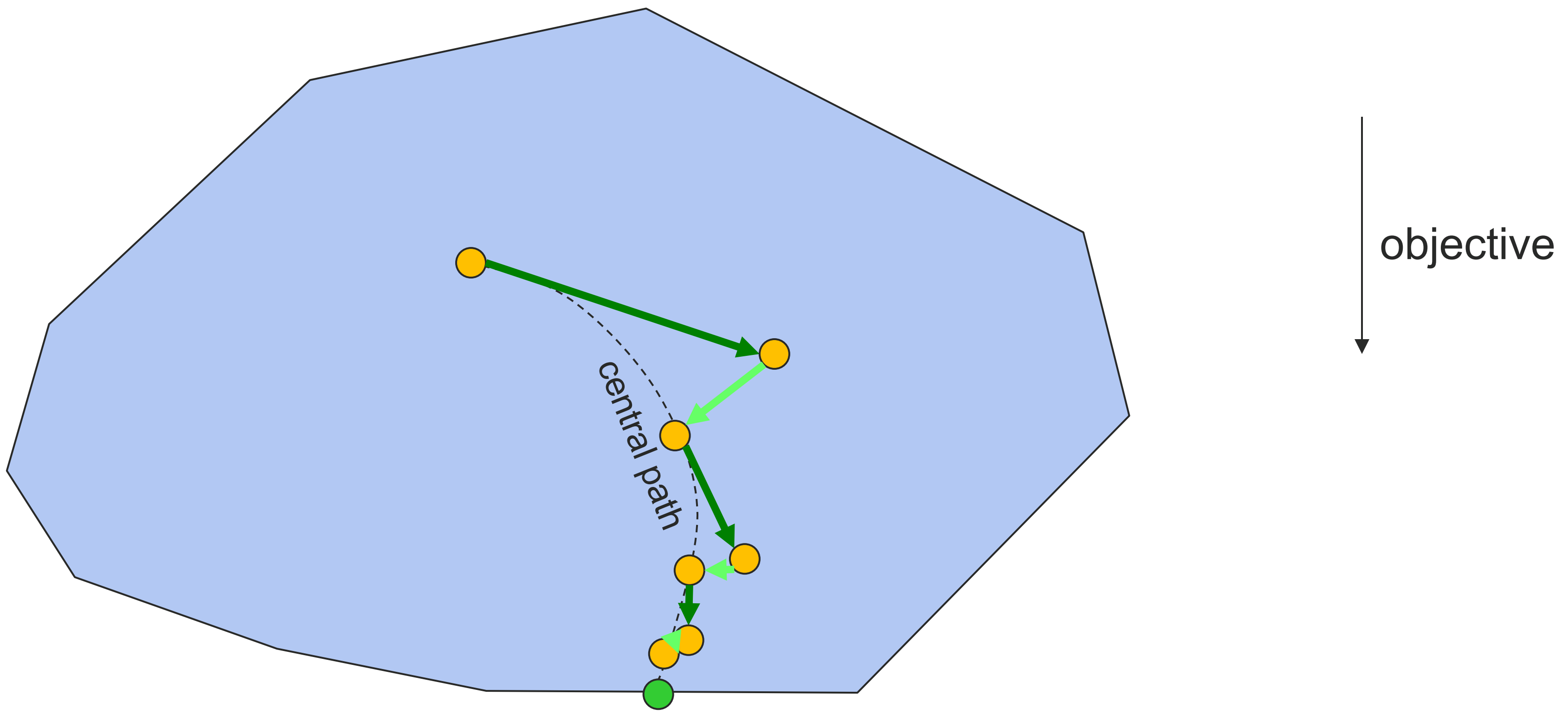


Simplex Log

```
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.7748600e+04   6.627132e+03   0.000000e+00      0s
    9643    1.1574611e+07   1.418653e+03   0.000000e+00      5s
   14440    1.1607748e+07   4.793500e+00   0.000000e+00     10s
   15213    1.1266396e+07   0.000000e+00   0.000000e+00     11s

Solved in 15213 iterations and 10.86 seconds
Optimal objective  1.126639605e+07
```

Interior Point Method



- Jump to the analytic center of the optimal face

Interior Point Method

- **Basic algorithm [Dikin, 1967, Karmarkar, 1984, Fiacco & McCormick, 1990]:**
  - Modify KKT conditions:

$$\begin{aligned} Ax &= b \\ A^T y + z &= c \\ x^T z &= 0 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} Ax &= b \\ A^T y + z &= c \quad (X = \mathrm{diag}(x)) \\ Xz &= \mu e \end{aligned}$$

  - Linearize complementarity condition:

$$\begin{pmatrix} -\theta & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} r_2 \\ r_1 \end{pmatrix} \qquad \text{(augmented system)}$$

$\theta_j = z_j / x_j, \quad x_j \cdot z_j = 0$ at optimality, so $\theta_j \rightarrow 0$ or ∞

  - Further simplification: $A\theta^{-1} A^T dy = b$ (normal equations)
  - Iterate, reducing μ in each iteration
  - Provable convergence

Interior Point Computational Steps

- **Setup steps:**
  - Presolve (same for simplex)
  - Compute fill-reducing ordering
  - Symbolic factorization – allocate static data structures for factor
- **In each iteration:**
  - Form $A\theta^{-1}A^T$
  - Factor $A\theta^{-1}A^T = LDL^T$ (Cholesky factorization)
  - Solve $LDL^T x = b$
  - A few Ax and $A^T x$ computations
  - A bunch of vector operations
- **Post-processing steps:**
  - Perform crossover to a basic solution
    - Optional, but typical
    - Especially when solving LP relaxations in a MIP solve

Barrier Log

```
Barrier statistics:
 AA' NZ     : 2.836e+03
 Factor NZ  : 3.551e+03 (roughly 40 MBytes of memory)
 Factor Ops : 1.739e+05 (less than 1 second per iteration)
 Threads    : 4

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   1.30273209e+06  0.00000000e+00  5.90e+02 0.00e+00  7.32e+00   12s
   1   1.04326180e+05 -5.84079103e+02  4.84e+01 1.69e+00  5.95e-01   12s
   2   9.46325157e+03 -4.40392705e+02  2.92e+00 1.35e+00  5.46e-02   12s
   3   3.66683689e+03  9.27381244e+02  1.94e-01 5.35e-01  1.41e-02   12s
   4   3.37449982e+03  1.79938013e+03  1.29e-01 2.41e-01  7.64e-03   12s
   5   3.13244138e+03  1.90266941e+03  8.89e-02 2.07e-01  6.00e-03   12s
   6   2.71282610e+03  2.11401255e+03  3.20e-02 1.15e-01  2.96e-03   12s
   7   2.48856811e+03  2.18107490e+03  1.06e-02 7.26e-02  1.56e-03   12s
   8   2.35427593e+03  2.21183615e+03  3.20e-03 4.52e-02  7.36e-04   12s
   9   2.30239737e+03  2.22464753e+03  1.53e-03 2.38e-02  4.03e-04   12s
  10   2.25547118e+03  2.23096162e+03  3.00e-04 1.40e-02  1.30e-04   12s
  11   2.24052450e+03  2.23917612e+03  4.10e-06 6.33e-04  7.20e-06   12s
  12   2.23967243e+03  2.23966346e+03  2.01e-08 5.01e-06  4.82e-08   12s
  13   2.23966667e+03  2.23966666e+03  1.11e-10 1.14e-13  4.81e-11   13s

Barrier solved model in 13 iterations and 12.51 seconds
Optimal objective 2.23966667e+03
```

Crossover Log

```
Barrier solved model in 13 iterations and 12.51 seconds
Optimal objective 2.23966667e+03


Root crossover log...

      40 DPushes remaining with DInf 0.0000000e+00                13s
       0 DPushes remaining with DInf 7.8159701e-14                13s

    1176 PPushes remaining with PInf 0.0000000e+00                13s
       0 PPushes remaining with PInf 0.0000000e+00                13s

  Push phase complete: Pinf 0.0000000e+00, Dinf 1.2079227e-13     13s


Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    1219    2.2396667e+03   0.000000e+00   0.000000e+00     13s
    1219    2.2396667e+03   0.000000e+00   0.000000e+00     13s

Root relaxation: objective 2.239667e+03, 1219 iterations, 0.43 seconds
```

Essential Differences for Parallelism

- **Simplex:**
  - Thousand/millions of iterations on extremely sparse matrices
  - Each iteration extremely cheap
  - Very limited opportunities to exploit parallel
- **Barrier:**
  - Dozens of expensive iterations
  - Much denser matrices
  - Lots of opportunities to exploit parallelism
    - But…

Concurrent LP

GUROBI OPTIMIZATION

- **Run both simplex and barrier simultaneously**
  - Thread 1: Dual simplex
  - Thread 2: Barrier
  - Thread 3: Barrier
  - Thread 4: Barrier
  - Thread 5: Primal simplex
  - Thread n≥6: Barrier
- **Solution is reported by first one to finish**
- **Use multiple CPU cores to exploit a diverse set of algorithms**
- **Best mix of speed and robustness**
- **Deterministic and non-deterministic versions available**

LP Performance

- **Performance results:**
  - Simplex on 1 core, barrier on all available cores
  - Concurrent:
    - 4 cores: 1 thread dual, 3 threads barrier
    - 16 cores: 1 thread primal, 1 thread dual, 14 threads barrier
  - Models that take >1s

4-core Xeon E3-1240

	GeoMean
Primal simplex	3.65
Dual simplex	2.25
Barrier	1.20
Concurrent	1.00
Deterministic Concurrent	1.13

16-core EPYC 7282

	GeoMean
Primal simplex	3.73
Dual simplex	2.56
Barrier	1.27
Concurrent	1.00
Deterministic Concurrent	1.20

Parallel Barrier Performance

Parallel Barrier Performance

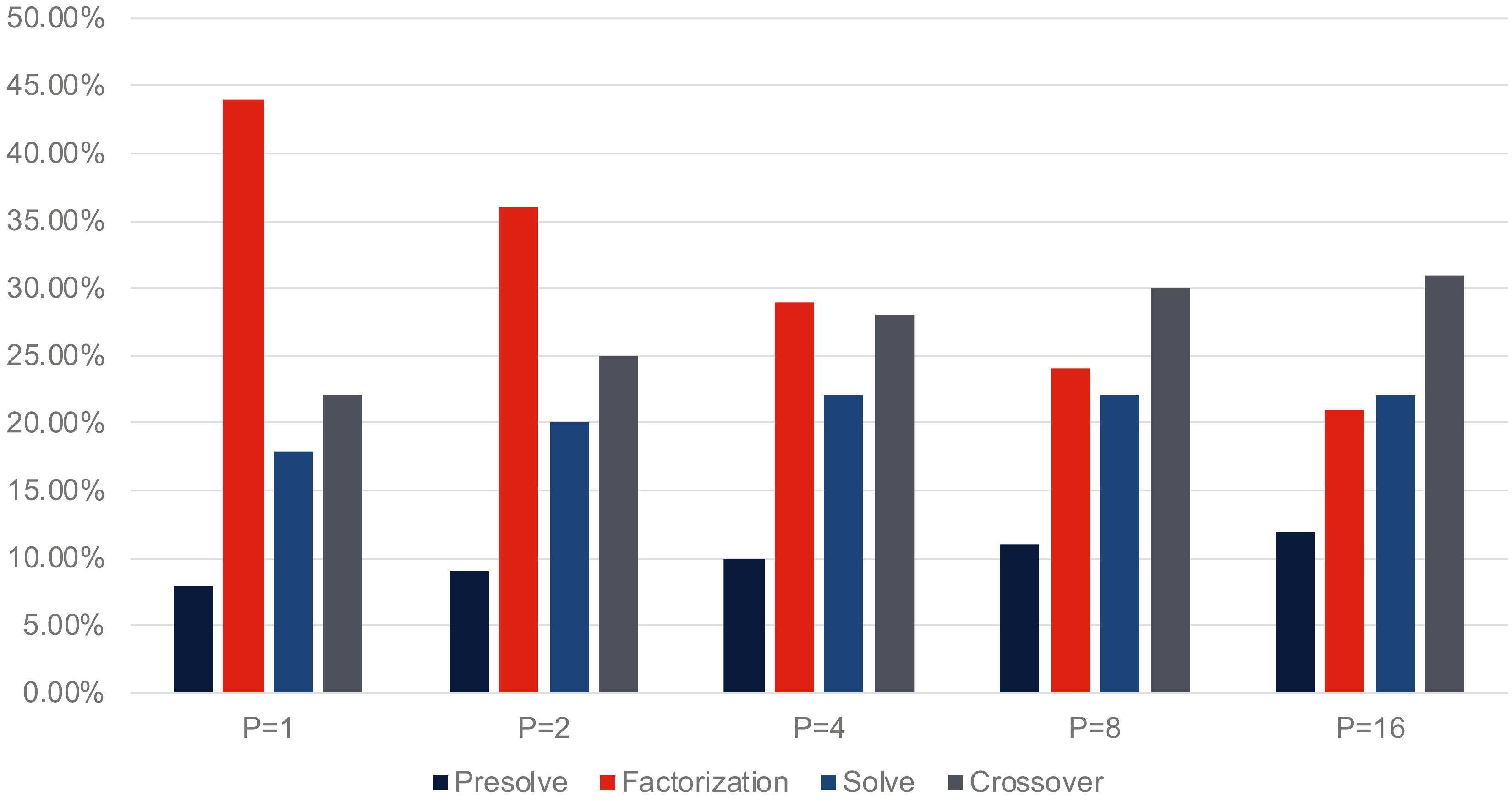
- **Speedups from adding cores**
  - On our internal LP test set
    - 1299 models
  - AMD EPYC 7282 (16 cores, 2.8GHz base, 3.2GHz boost, 64GB 2933MHz DDR4)
  - Relative to one core

	# models	P=1	P=2	P=4	P=8	P=16
>1s	602	1.00	1.28	1.54	1.76	1.92
>10s	430	1.00	1.34	1.69	1.97	2.20
>100s	249	1.00	1.46	1.86	2.22	2.56

Tests run 2020-07-18

Barrier Runtime Breakdown

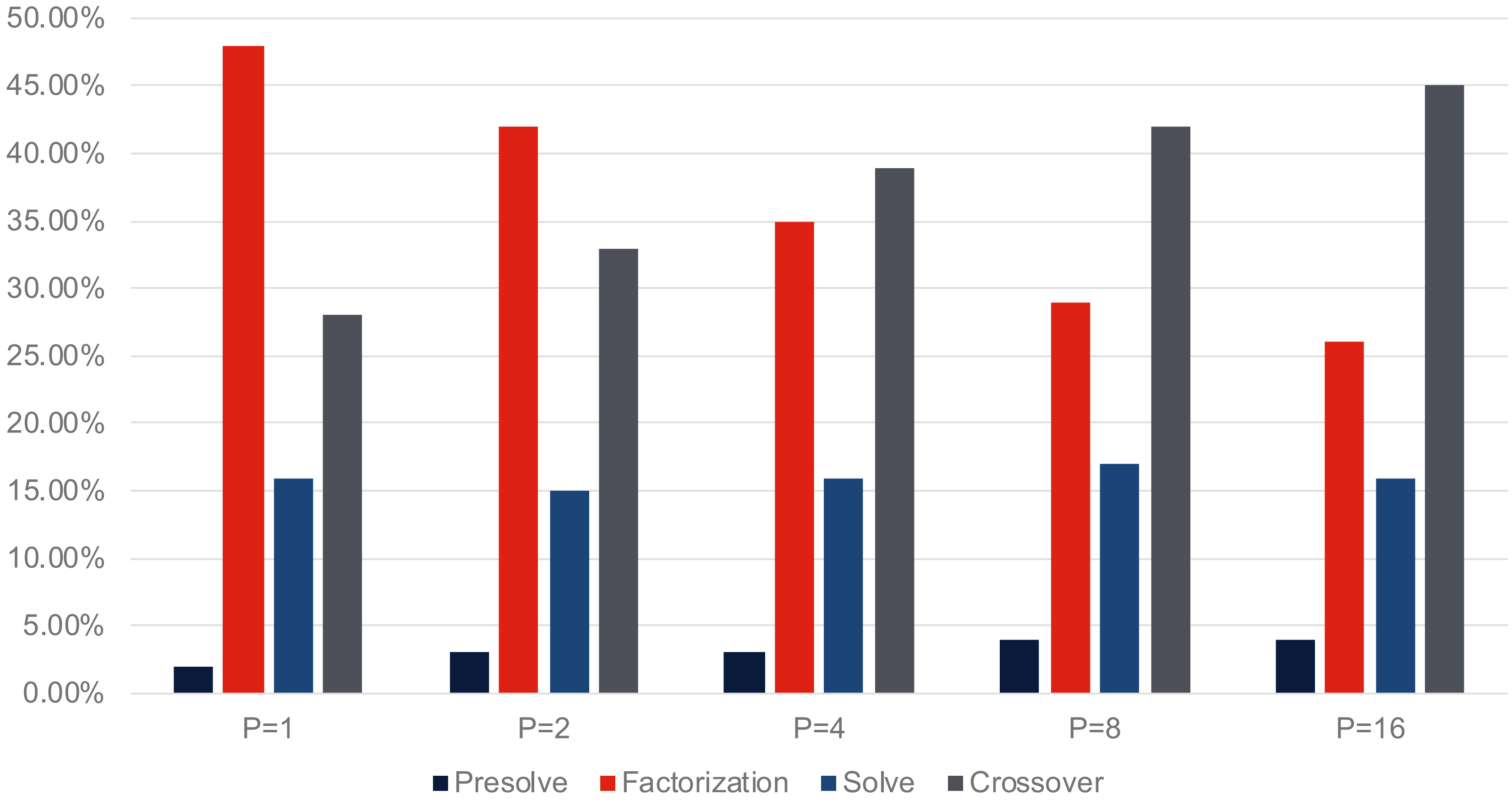
- Fraction of total runtime (>1s, 602 models)



Tests run 2020-07-18

Barrier Runtime Breakdown

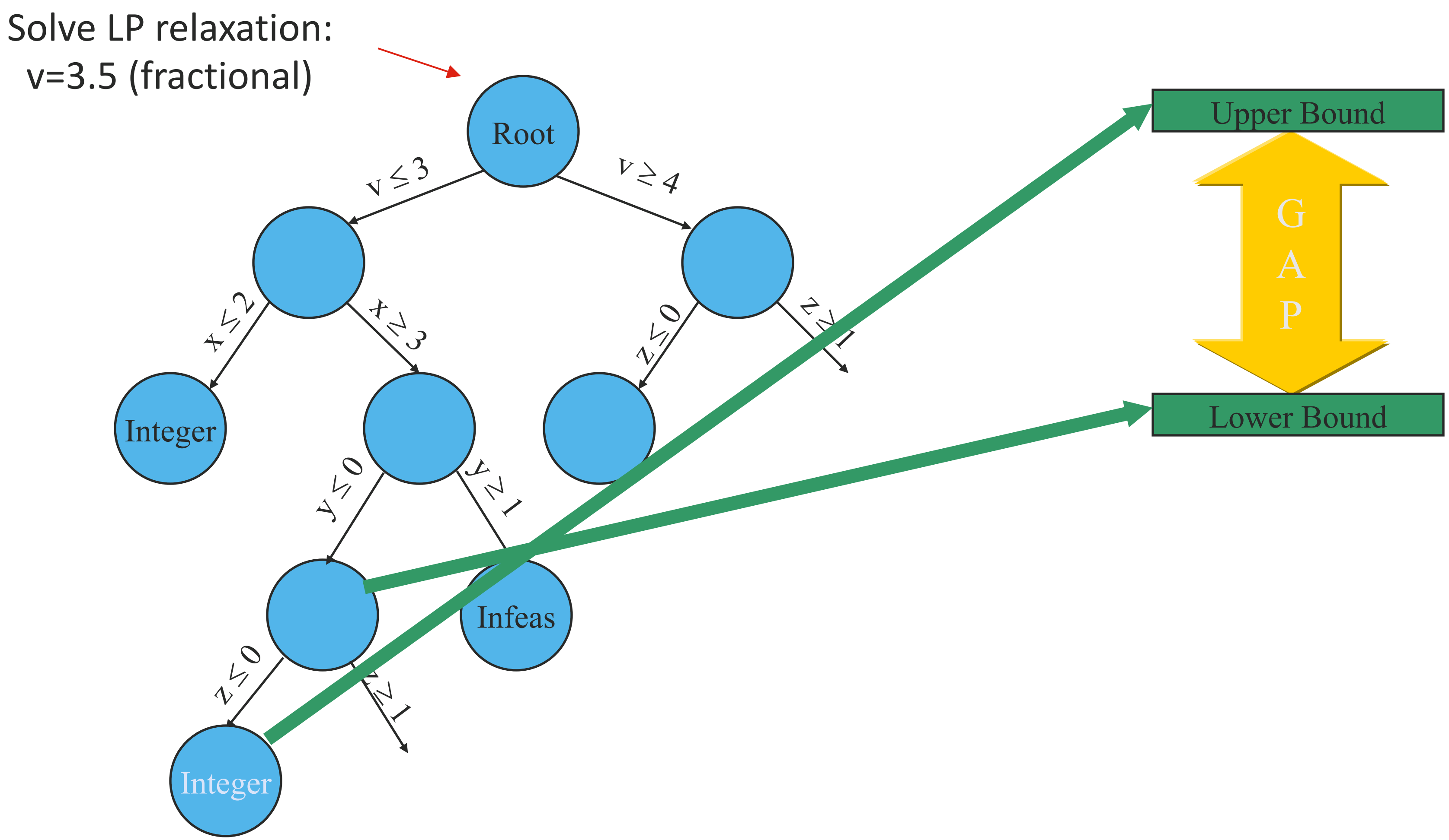
- Fraction of total runtime (>100s, 249 models)



Tests run 2020-07-18

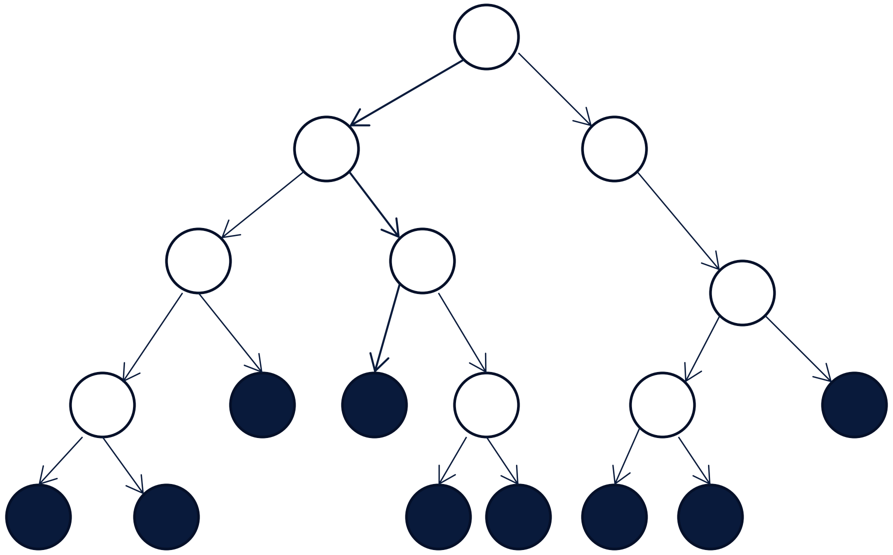
Parallelism in MIP

MIP Solution Framework: LP-based Branch-and-Bound



Parallel MIP = Parallel Branch and Bound

- MIP explores a tree of relaxations



- Frontier nodes are independent and can be explored in parallel

Parallel MIP Performance

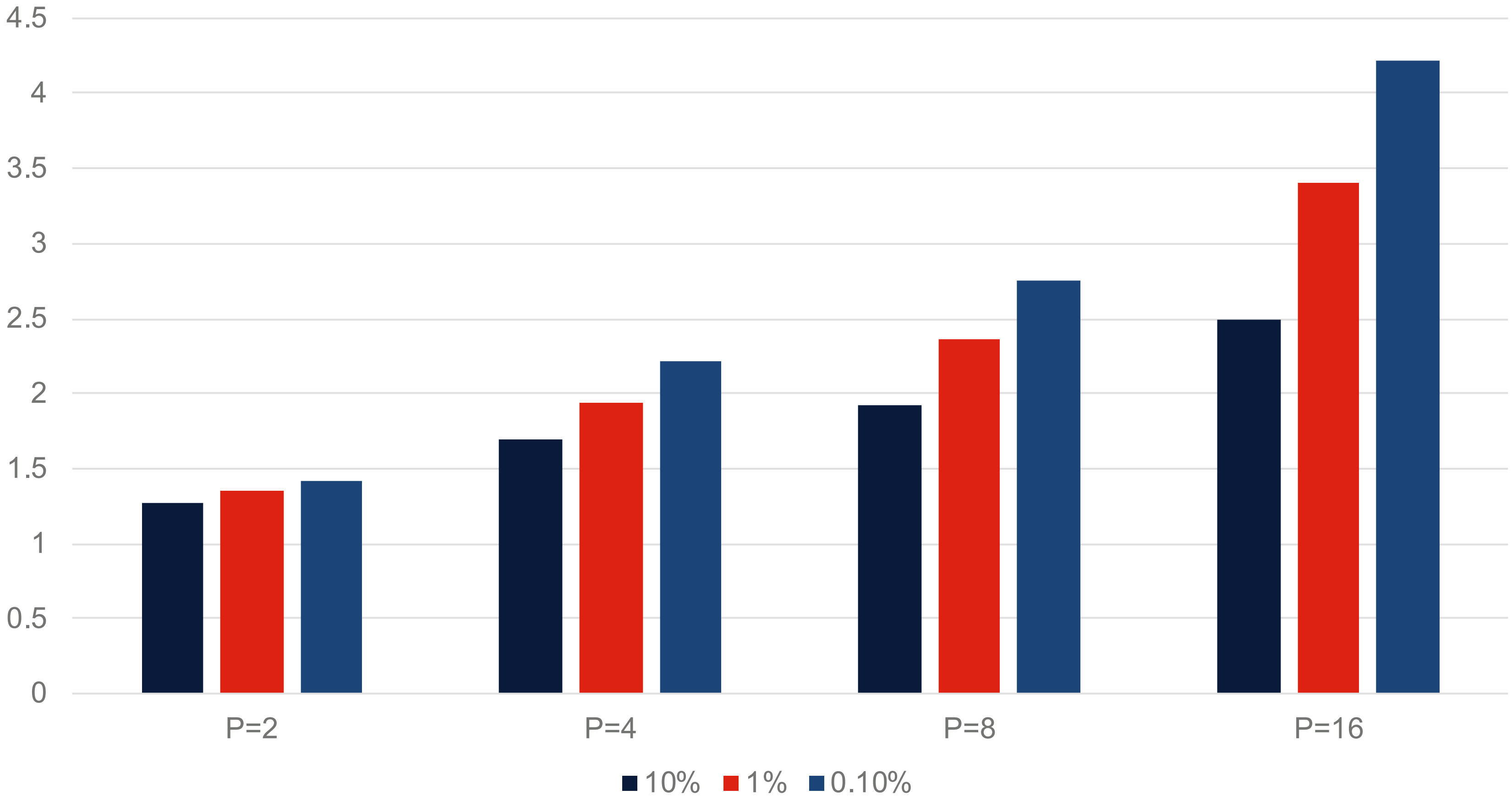
- Speedups from adding cores
  - On our internal MIP test set
    - 3965 models
  - AMD EPYC 7282 (16 cores, 2.8GHz base, 3.2GHz boost, 64GB 2933MHz DDR4)
  - Relative to one core

	# models	P=1	P=2	P=4	P=8	P=16
>1s	2654	1.00	1.26	1.72	1.96	2.11
>10s	1907	1.00	1.32	1.91	2.24	2.47
>100s	1087	1.00	1.42	2.21	2.74	3.08
>1000s	319	1.00	1.68	3.03	3.86	4.22

Tests run 2020-07-20

Parallel MIP Performance Versus Optimality Gap

- Parallel speedup versus optimality gap (>100s – 1087 models)



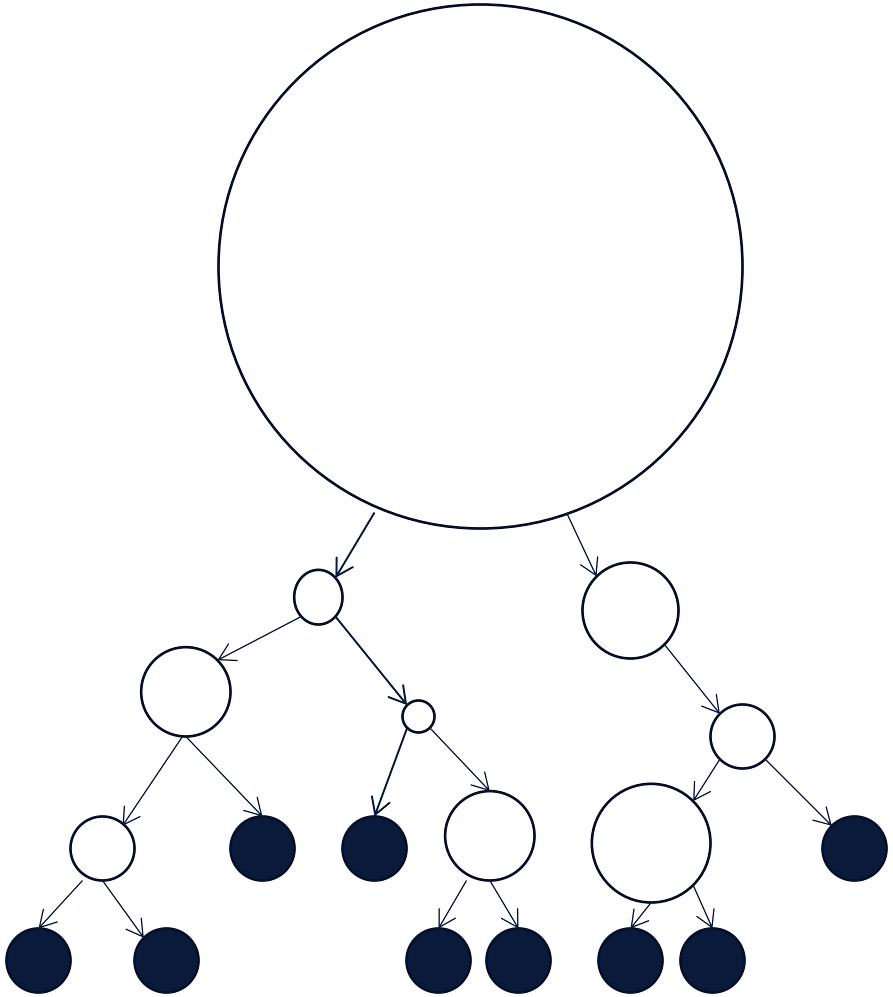
Tests run 2020-07-20

What Limits MIP Parallelism?

- **Tree shape**
  - Fraction of time spent at the root
  - Total number of nodes explored
  - Load balancing
  - Topology of the tree

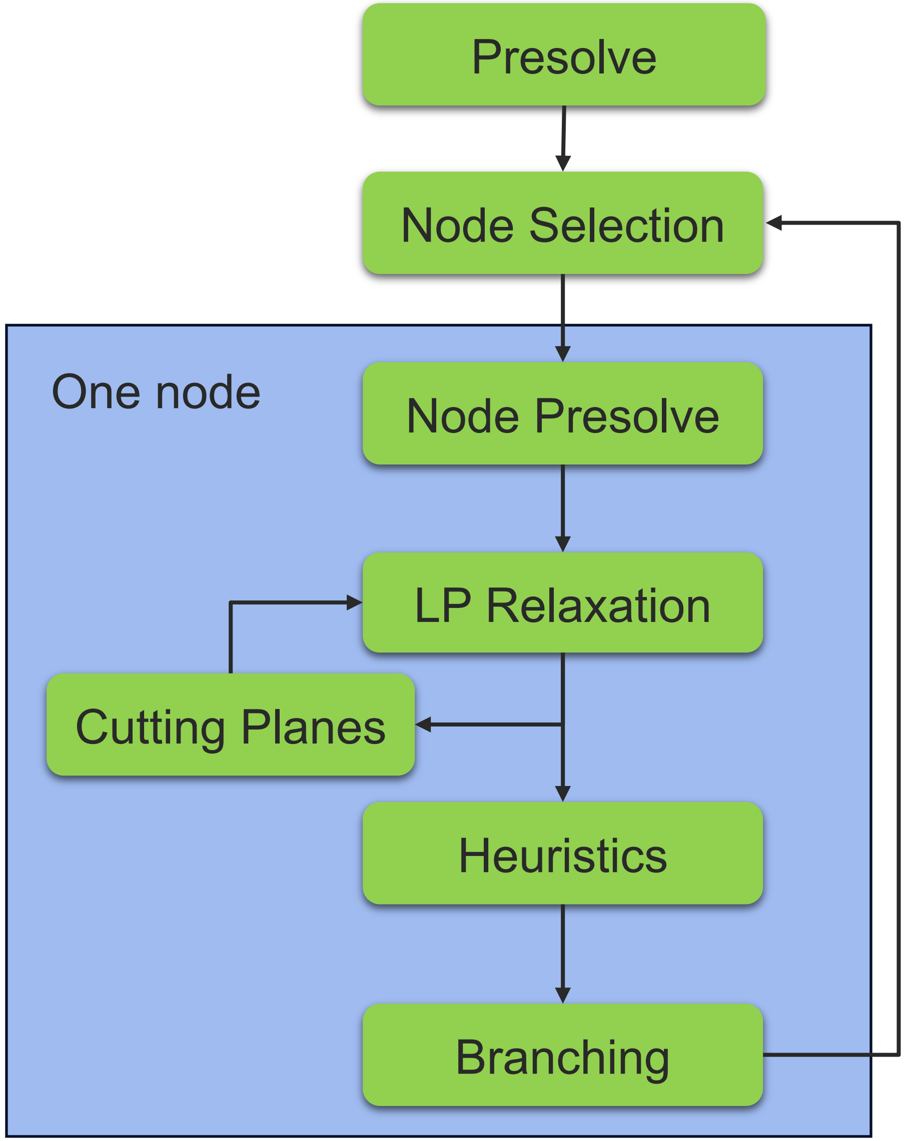
What Limits MIP Parallelism?

- **Tree shape**
  - Fraction of time spent at the root
  - Total number of nodes explored
  - Load balancing
  - Topology of the tree



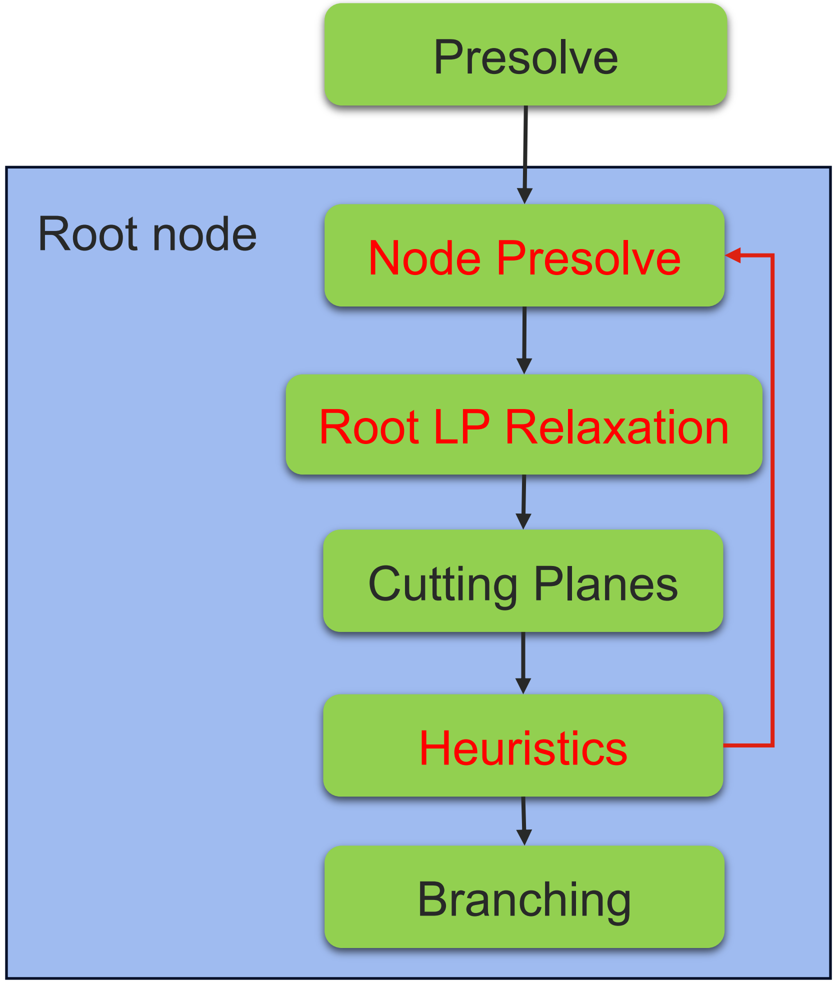
What Happens at a Node?

- **Multiple steps at each node**
  - Node presolve
  - LP relaxation solve
  - Cutting planes
  - Heuristics
  - Branch variable selection



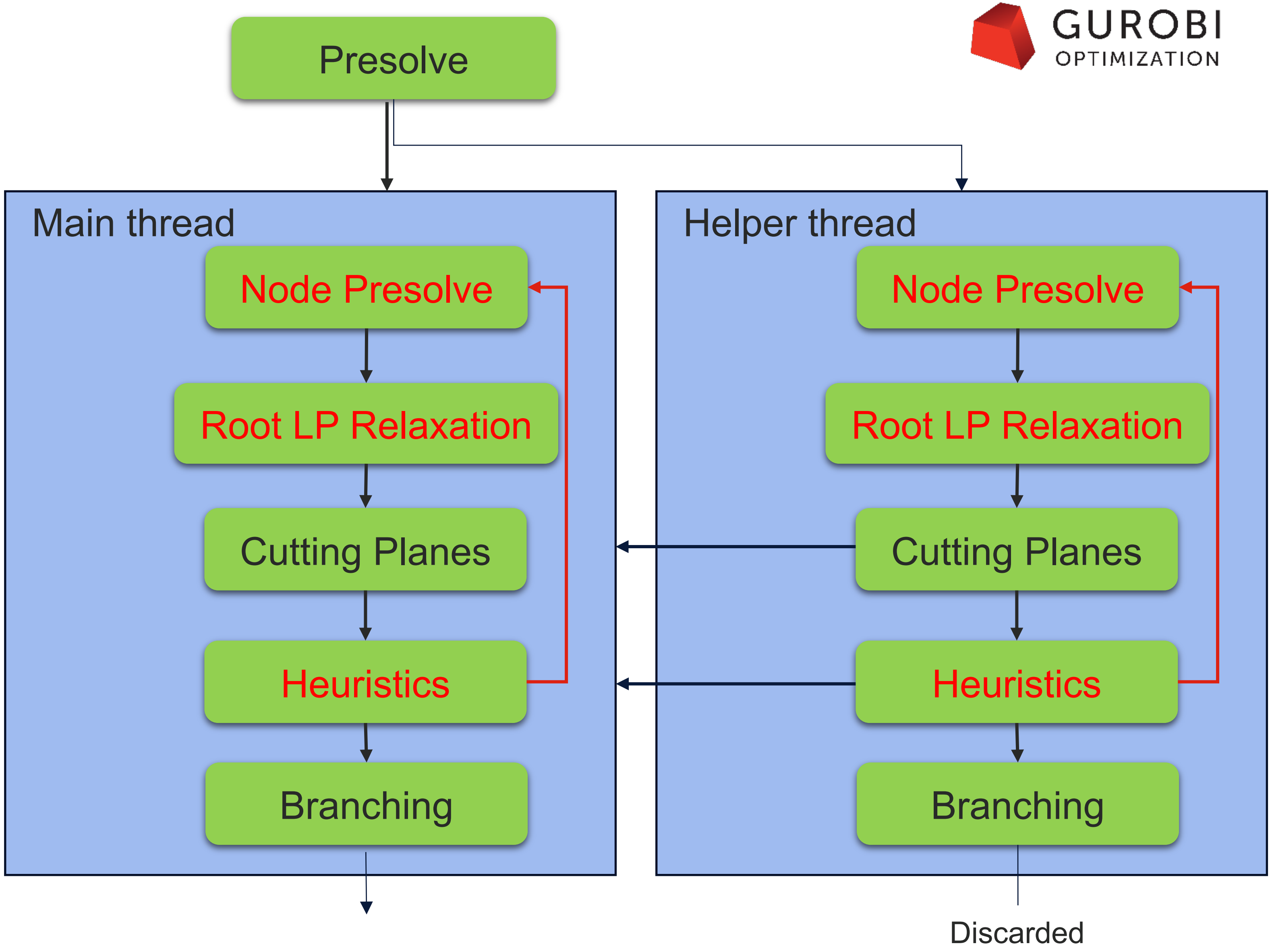
What Happens at the Root?

- Root node repeats these steps many times
  - 10+ passes not unusual
- Vital to make as much progress as possible before branching



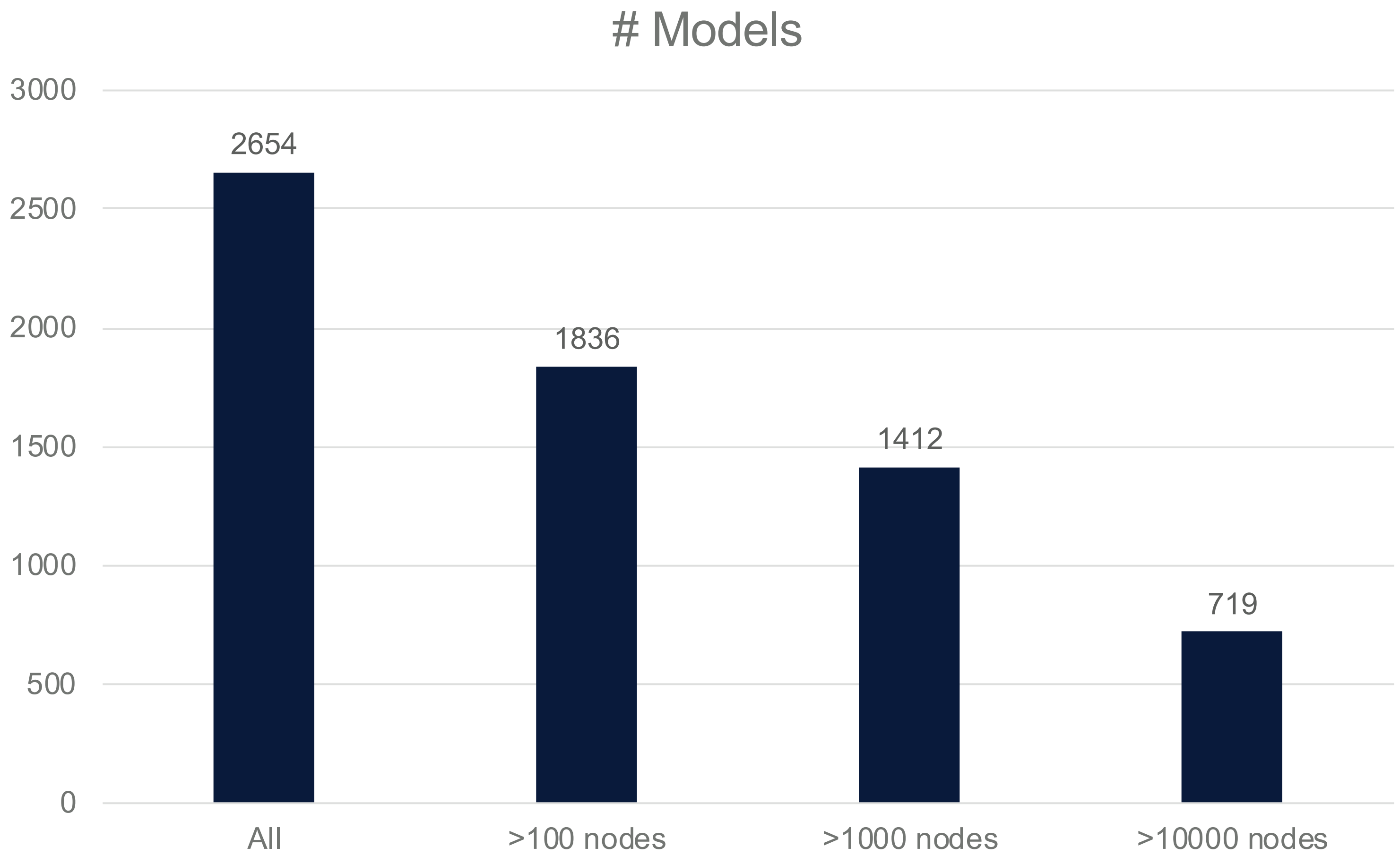
Parallelism at the Root

- **Options for exploiting multiple cores at the root?**
- **Exploit performance variability [Fischetti, Lodi, Monaci, Salvagnin, 2014]**
  - Start one or more *helper* threads
  - Same steps as main thread, but perturbed slightly
  - Feed results back to main thread
    - Heuristic solutions
    - Cutting planes
- **Limited benefit**



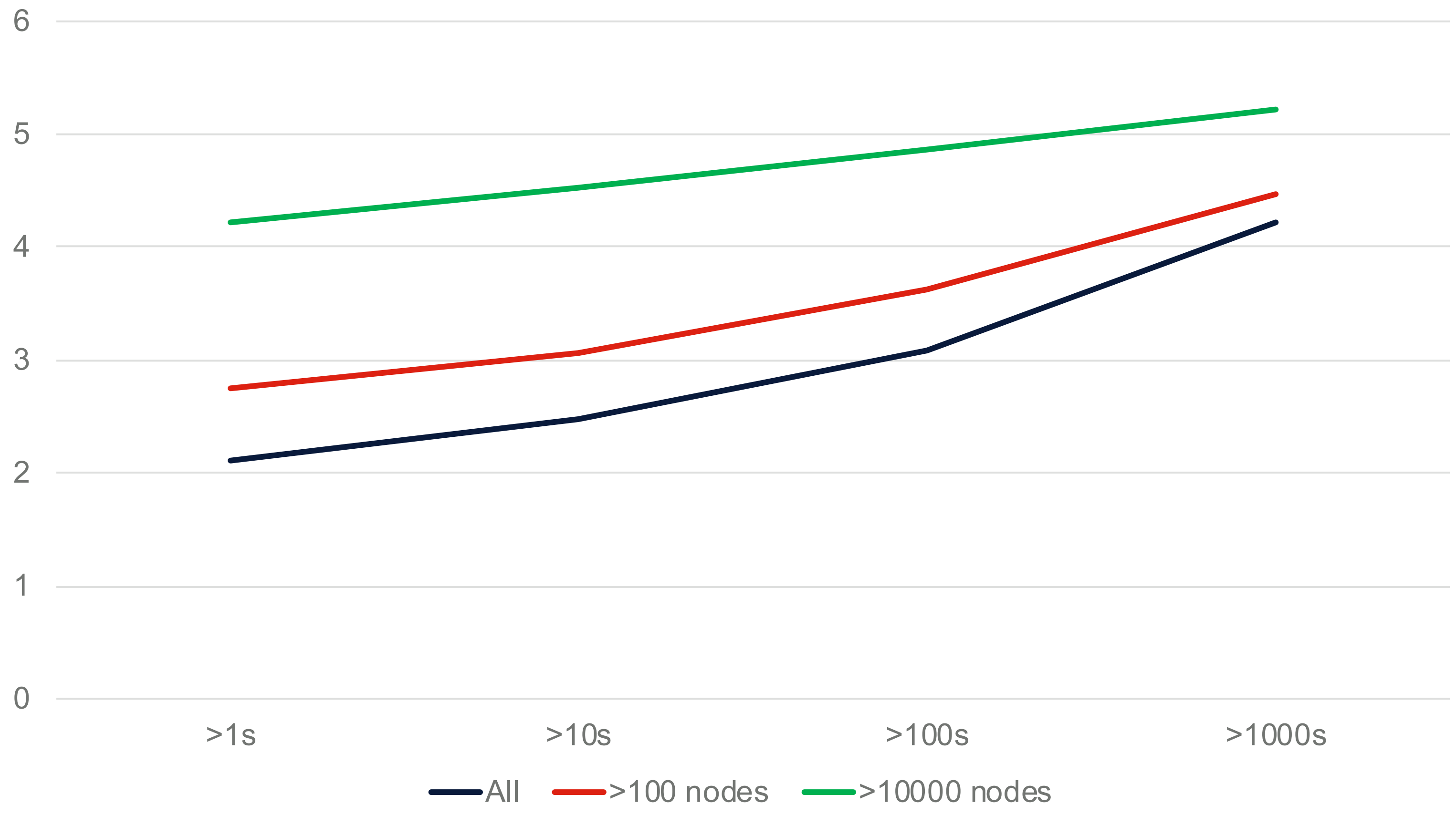
What Limits MIP Parallelism?

- **Tree shape**
  - Fraction of time spent at the root
  - Total number of nodes explored
  - Load balancing
  - Topology of the tree



Parallel MIP Performance (By Node Count)

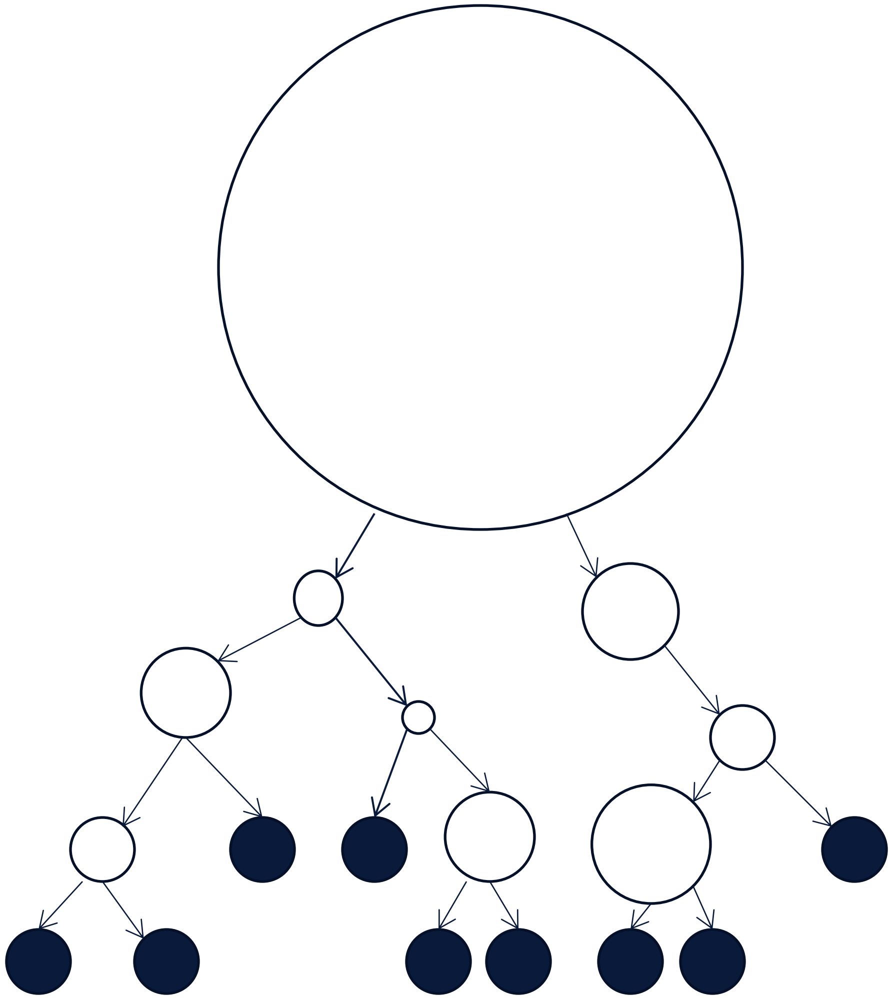
- Geometric mean speedup on 16 cores



Tests run 2020-07-20

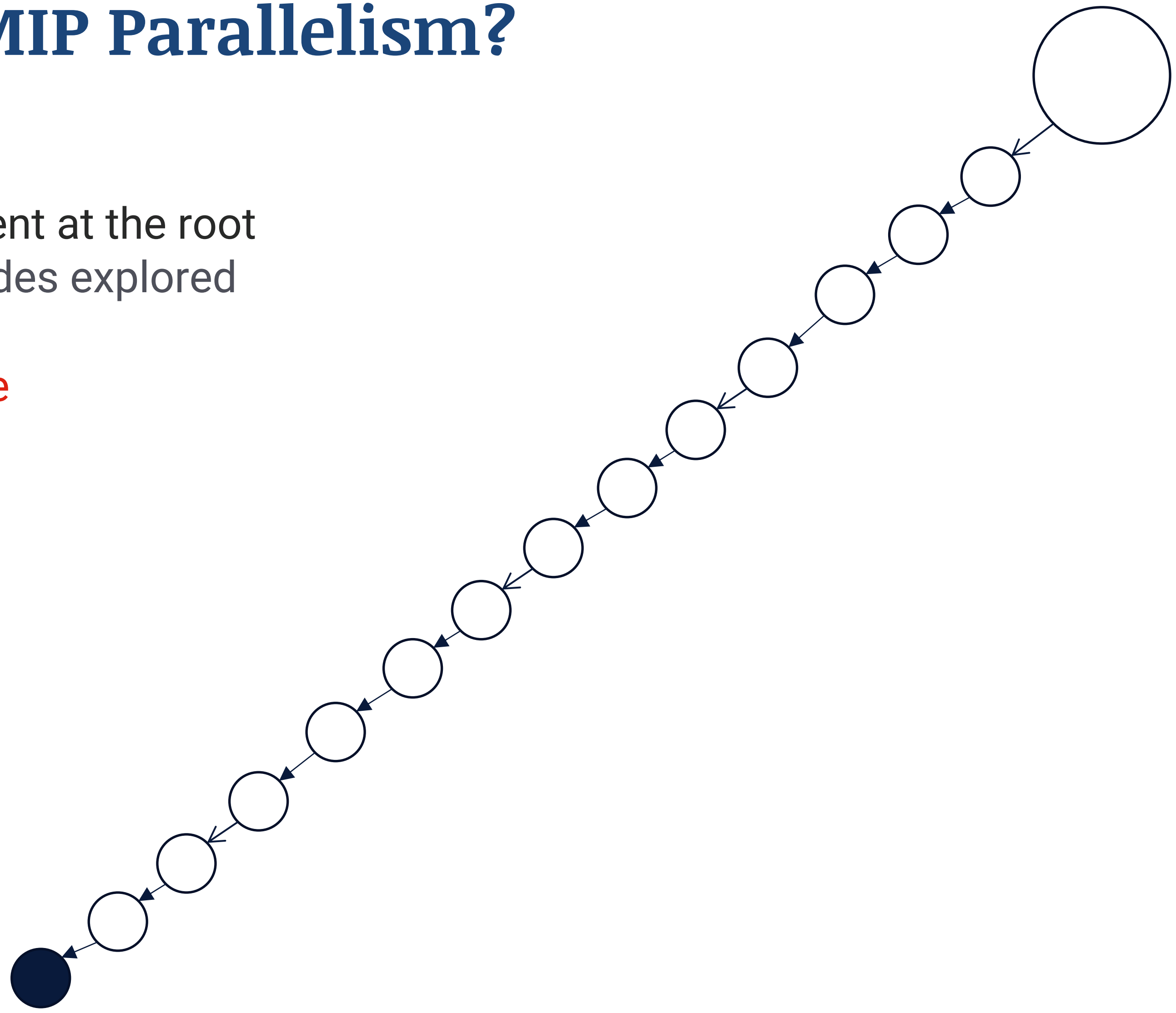
What Limits MIP Parallelism?

- **Tree shape**
  - Fraction of time spent at the root
  - Total number of nodes explored
  - Load balancing
  - Topology of the tree



What Limits MIP Parallelism?

- **Tree shape**
  - Fraction of time spent at the root
  - Total number of nodes explored
  - Load balancing
  - Topology of the tree



Other Options

Concurrent MIP

- **Use `ConcurrentMIP=n` parameter**
  - `n` independent MIP jobs
    - Divide available threads among requested jobs
      - Example: 24 threads available, `ConcurrentMIP=4`: 4 jobs, 6 threads each
    - Results combined automatically
    - Settings
      - Default: different random seeds
      - User can control with *concurrent environments*
  - *Non-deterministic*
- **Scope for improvement**
  - Exploit unpredictability
    - Best (random) result wins
  - Focus on different goals simultaneously
    - E.g., one run works on lower bound (cuts, etc.), one works on upper bound (heuristics, etc.)
- **Not as effective as you might hope in general**
  - Some notable exceptions

Concurrent MIP – How To

- Simplest approach: use `ConcurrentSettings` command-line parameter
  - MIPFocus1.prm: `MIPFocus 1` (focus on feasible solutions)
  - MIPFocus3.prm: `MIPFocus 3` (focus on lower bound)
- Result for model `dg012142…`

ConcurrentSettings=MipFocus1.prm,MipFocus3.prm

```
 Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
…
    0     2     -    -    -    6492675.78 778732.766  88.0%     -    8s
   83    92     -    -    -    5148647.37 785662.888  84.7%     -   10s
  149   156     -    -    -    3044594.17 785662.888  74.2%     -   19s
  180   187     -    -    -    2984859.29 785662.888  73.7%     -   22s
  188   195     -    -    -    2727098.75 836163.745  69.3%     -   26s
  196   233     -    -    -    2720275.25 859256.557  68.4%     -   36s
```

Default settings

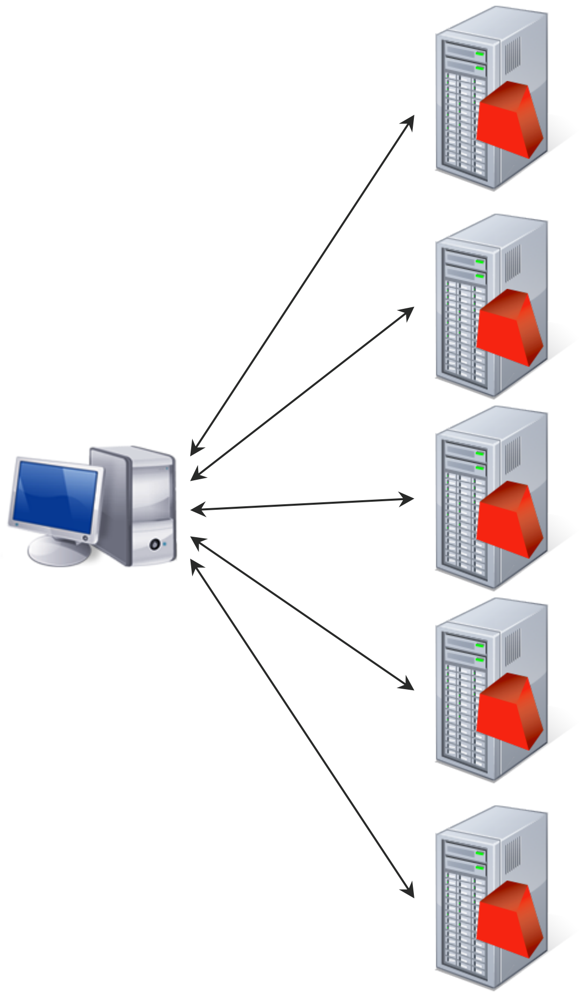
```
 Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
…
     0     2 772593.572    0  410 3.7336e+07 772593.572  97.9%     -    2s
H  126   129                    3.181386e+07 775156.902  97.6%   230    3s
H  365   374                    2.537530e+07 775156.902  96.9%   195    3s
   623   651 1484472.12   29  330 2.5375e+07 775156.902  96.9%   193    5s
H  625   651                    2.228999e+07 775156.902  96.5%   192    5s
H  635   651                    1.700295e+07 775156.902  95.4%   192    5s
H  649   651                    1.524965e+07 775156.902  94.9%   191    5s
H 1309  1146                    5265663.3333 775156.902  85.3%   162    9s
H 1312  1144                    5032846.8222 775156.902  84.6%   162    9s
H 1316  1123                    4228610.0476 775156.902  81.7%   162    9s
H 1320  1107                    3836260.5089 775156.902  79.8%   162    9s
H 1322  1101                    3628016.2000 775156.902  78.6%   163    9s
H 1346  1101                    3575756.5000 775156.902  78.3%   162    9s
  1530  1212 782874.062    6  367 3575756.50 775636.708  78.3%   155   10s
  2466  1926 1301398.44   29  410 3575756.50 775636.708  78.3%   144   17s
  2483  1937 3018571.09   72  425 3575756.50 775686.218  78.3%   143   20s
  2513  1960 779009.893   12  424 3575756.50 779009.893  78.2%   146   29s
  2535  1983 2621748.16   16  340 3575756.50 780507.097  78.2%   148   30s
```

Distributed Algorithms

Distributed Architecture

- **Multiple, ideally identical machines connected by a network**
  - On premise
  - Gurobi Instant Cloud

- **Manager-worker paradigm**
  - Manager distributes work among workers
  - Workers perform work and report results
  - Manager collects results



Distributed MIP

- Use `DistributedMIPJobs=n` parameter
- Actually a combination of concurrent and parallel tree exploration
  - Ramp up: concurrent for a limited number of nodes [*ParaSCIP, 2010*]
  - Parallel tree exploration: continue with the ‘best’ concurrent result
- Dramatically higher node throughput…
  - …when the search tree makes lots of independent nodes available

Model	1 Machine	16 Machines	32 Machines
danoint	1933s 912K nodes	196s 9.9X faster	128s 15.1X faster

- Easy to try using Gurobi Instant Cloud

Gurobi Version 9.0.3
Intel Xeon E3-1240 v3 CPUs

Distributed Tuning

- Automatic parameter tuning
- Use `TuneJobs=n` parameter for distributed parallel tuning
- Trivially parallel
  - Explore different parameter settings in parallel
- Typical to get linear parallel speedups
  - With 24 machines, a day becomes an hour

Other Metrics

Thread Control on a Shared Machine

- **Imagine multiple optimization jobs share a single machine**
- **How many threads should each one use?**
  - Too few: leave cores idle
  - Too many: multiple jobs fight over cores

Thread Control on a Shared Machine

- Instead of measuring completion time for one model, measure *machine throughput*
  - Number of times model `danoint` can be solved in our hour
  - Running 1, 2, 4, 8, 16 or 16 jobs simultaneously, using different per-job core counts
  - 24-core Intel Xeon Gold 5118, 2.3GHz, 512GB DDR3 system

Threads per job	1 Job	2 Jobs	4 Jobs	8 Jobs	16 Jobs
12	15.0	27.7	38.9	38.4	38.0
19				38.0	38.4
24	27.7	40.7	40.0	39.9	38.9

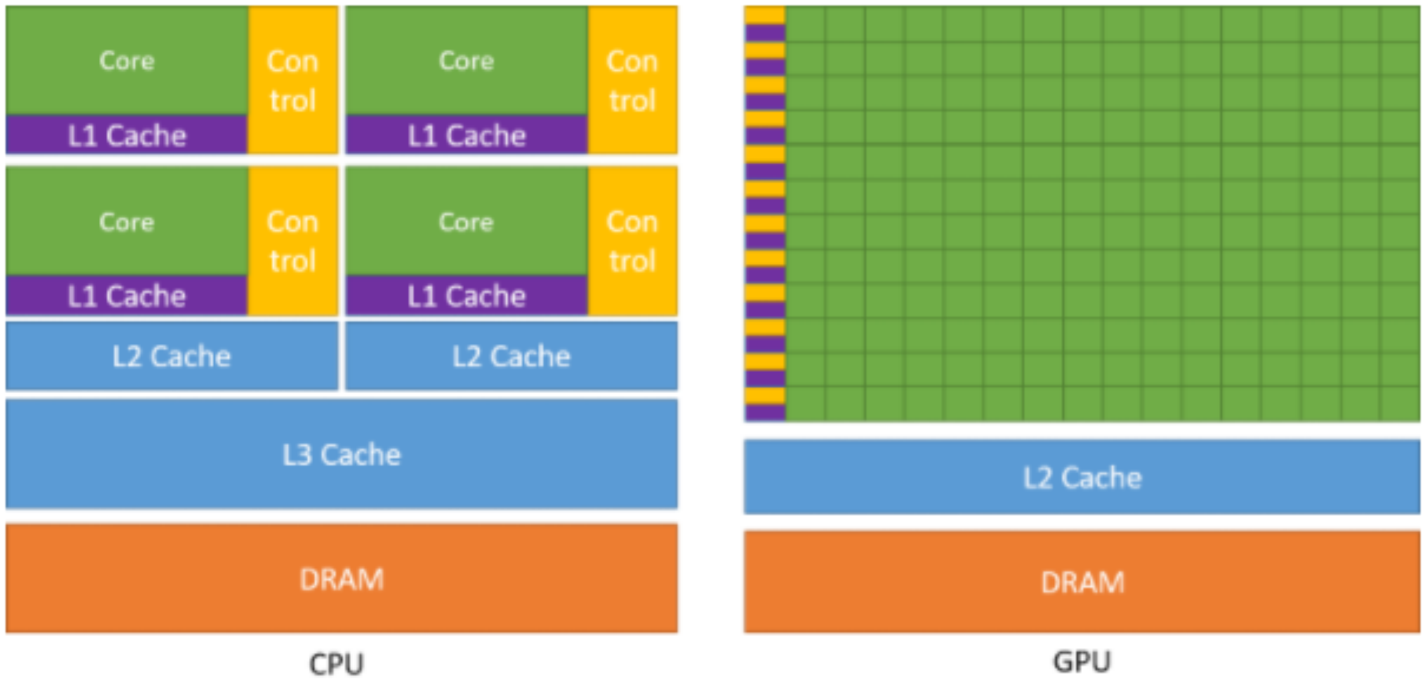
- No need to worry about matching thread count to core count

Tests run 2020-08-08

Other Architectures

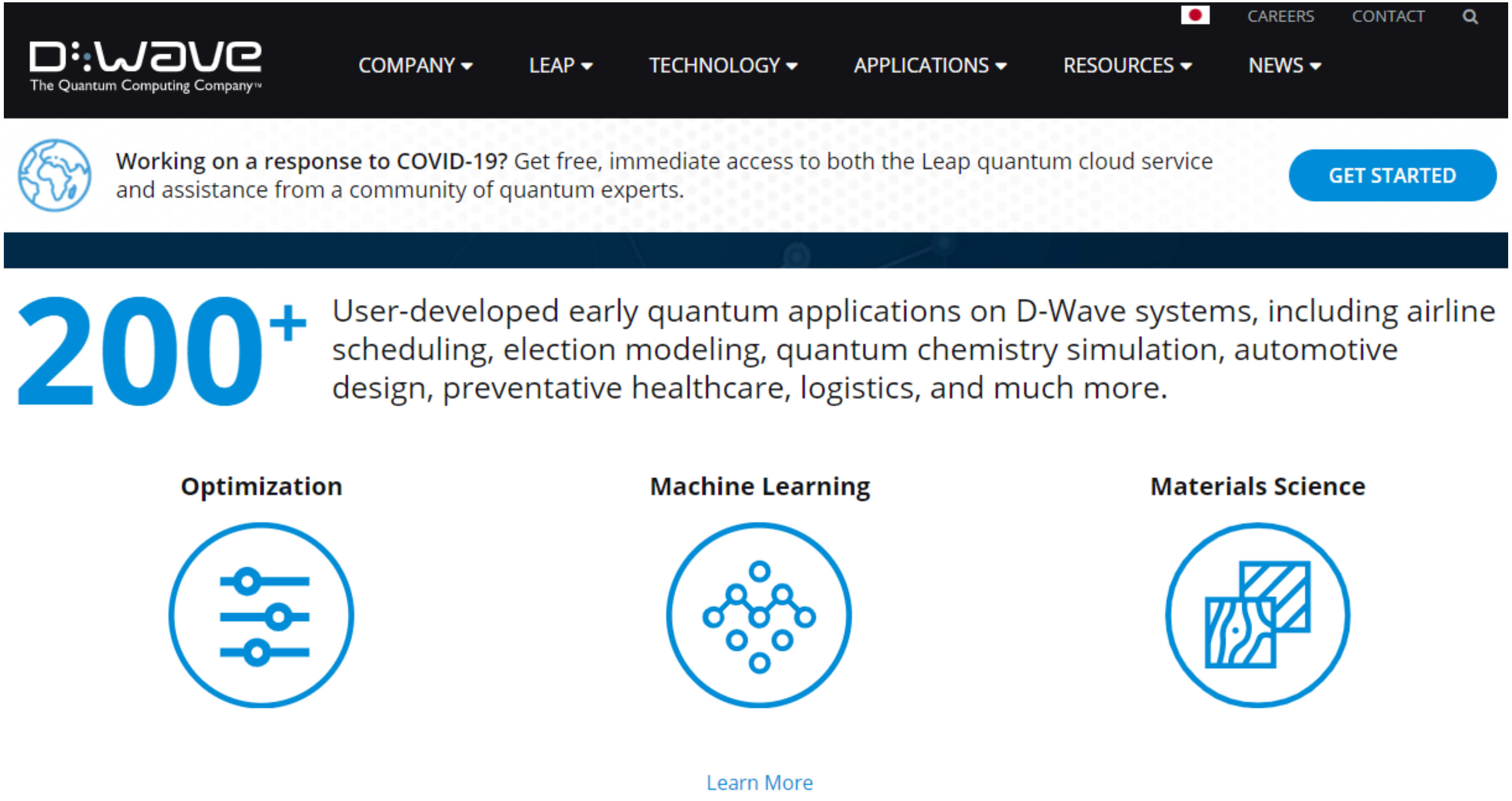
Graphical Processing Unit (GPU) Computing

- Single-Instruction Multiple-Data (SIMD) Computing



Copyright 2020, NVIDIA Corporation

Quantum Computing



Quantum Computing

- Interesting future technology
- Potential to substantially speed up optimization tasks
- Currently still a science project

Conclusions

- **Parallelism used throughout the Gurobi Optimizer**
  - LP (barrier and concurrent)
  - MIP
  - Distributed MIP
  - Distributed tuning
- **Significant performance improvements in most cases**
  - Not linear
  - Problem dependent
- **Continued focus area**
  - Parallelism continues to become more important

Thank You – Questions?

GUROBI OPTIMIZATION

The World's Fastest Solver

Your Next Steps

- Try Gurobi 9.0 Now!
  - Get a 30-day commercial trial license of Gurobi at www.gurobi.com/free-trial
  - Academic and research licenses are free!
- For questions about Gurobi pricing, please contact sales@gurobi.com or sales@gurobi.de
- A recording of this webinar, including the slides, will be available in roughly one week.