The Gurobi Optimizer

GUROBI
OPTIMIZATION

Gurobi History

- Gurobi Optimization, founded July 2008
  - Zonghao Gu, Ed Rothberg, Bob Bixby
  - Started code development March 2008
- Gurobi Version 1.0 released May 2009
- History of rapid, significant performance improvements
  - Close to 2x average speedup year-over-year
- Over 1200 companies have become Gurobi customers
  - Performance, superior support, transparent pricing and licensing
- History of continuing, significant innovations
  - Free academic licenses
  - First cloud offering
  - Compute-server for client-server applications
  - Distributed algorithms

Gurobi Algorithms

- Mixed-Integer Programming (MIP, MIQP, MIQCP)
  - LP based branch-and-cut
  - Parallel
  - Concurrent
  - Distributed concurrent
  - Distributed parallel
- Linear and Quadratic Programming
  - Primal simplex
  - Dual simplex
  - Parallel Barrier
  - Concurrent
  - Distributed concurrent
- Second-Order Cone Programming
  - Parallel Barrier
- Linear Programming with piecewise linear objective
  - Primal simplex
  - Dual simplex

Gurobi APIs

- C, C++, Java, .NET, Python programming interfaces

- Simple command-line interface
- Python interactive interface
- Python modeling interface

- R and MATLAB matrix interfaces
- All standard modeling languages
- A variety of free-software projects

Gurobi Products

- Commercial and Academic Licenses
  - Licensing options and pricing available at www.gurobi.com
  - Free for academic use
  - "Take Gurobi With You" program
- Cloud offering
  - Amazon EC2
  - Gurobi Instant Cloud
- Gurobi Compute Server

The Gurobi Amazon Cloud

- Optimize on as many machines as you need, when you need them
- No need to purchase new computers or new Gurobi licenses
- Pay for what you use – only pay for software and computers when you need to solve optimization models
- Distributed computing is included at no extra charge except for the cost of the machines
- No change is required to your code
- Use it alone or to supplement licenses you already own
- Available on Amazon Web Services

Gurobi Instant Cloud

- Gurobi Instant Cloud makes it fast and easy to use the cloud for optimization.
- Simply install Gurobi Optimizer on your computers, then connect them to the cloud in one step:
  - Launch the Gurobi Instant Cloud from your account on www.gurobi.com
  - Select a nearby data center and fast computers to quickly solve your models
  - Use the Gurobi Instant Cloud dashboard to start and stop cloud computers and to configure your client computers
  - Access it over the Internet via any Windows, Linux or Mac computer
  - Confidently connect to your cloud server using automatic
    - 256-bit AES encryption

How To Get Started Using Gurobi

- Gurobi 6.0 is installed on your virtual machine
  - The license key expires October 31
  - Easy to get a free academic license key on Gurobi website

- Various examples in all programming APIs can be found in the "examples" sub-directory of the Gurobi distribution

- Some rich examples with detailed problem description and interactive demo are available on our website
  - http://examples.gurobi.com

- Let's briefly look at one of them
  - http://examples.gurobi.com/FCNFexample/

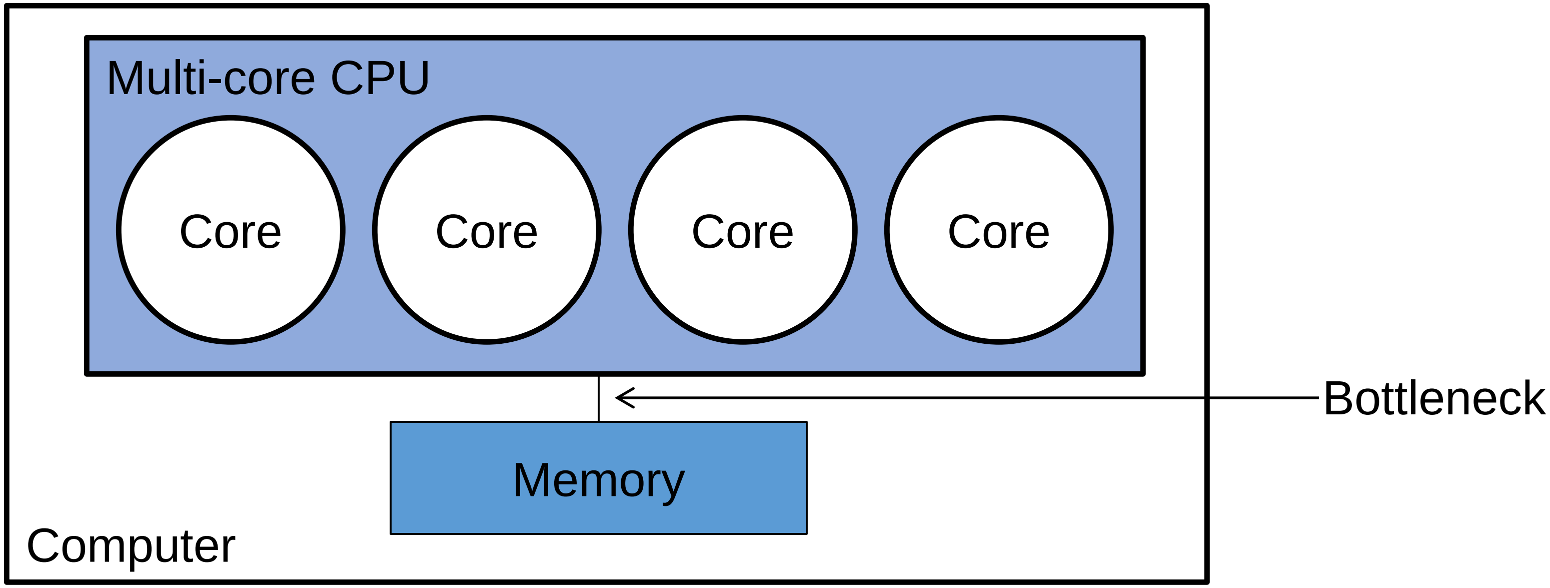
Distributed Optimization

Parallel algorithms and hardware

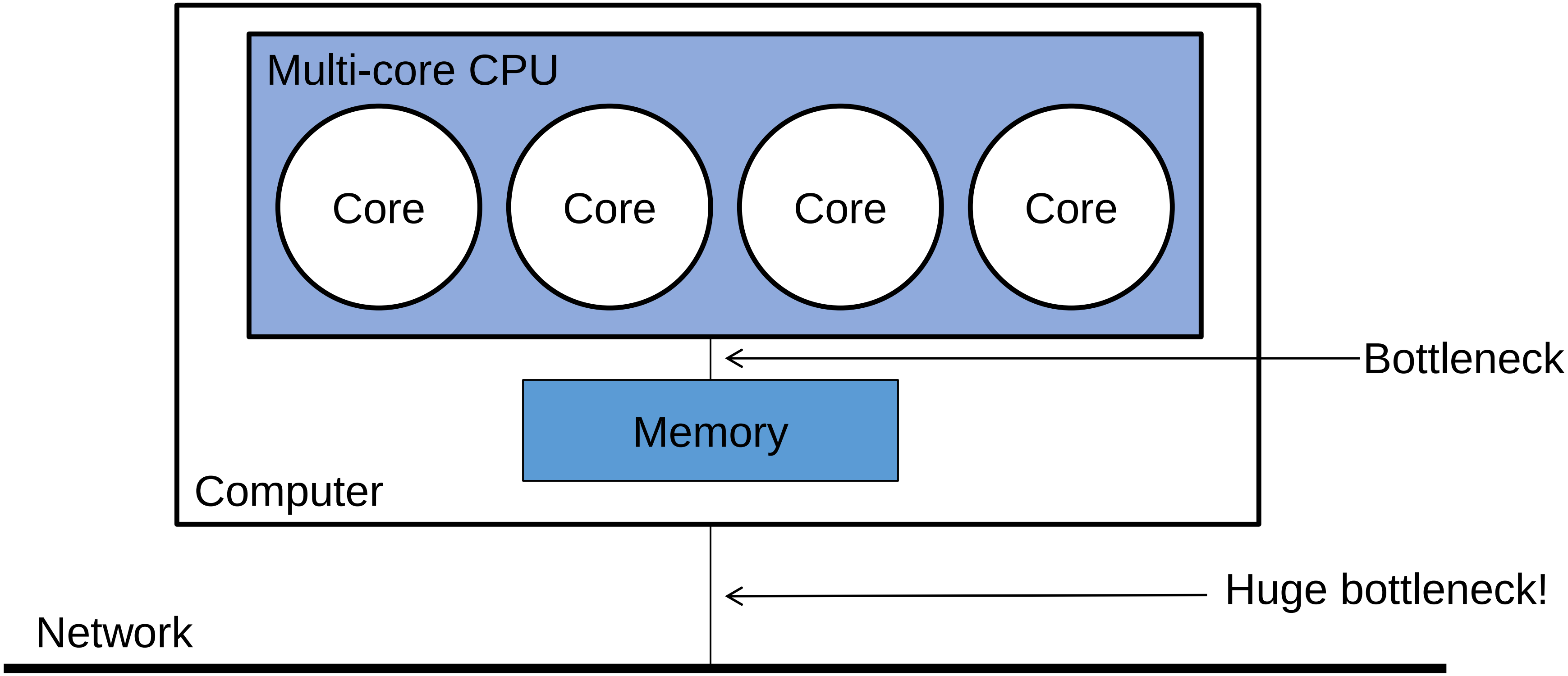
- Parallel algorithms must be designed around hardware
  - What work should be done in parallel
  - How much communication is required
  - How long will communication take

- Goal: Make best use of available processor cores

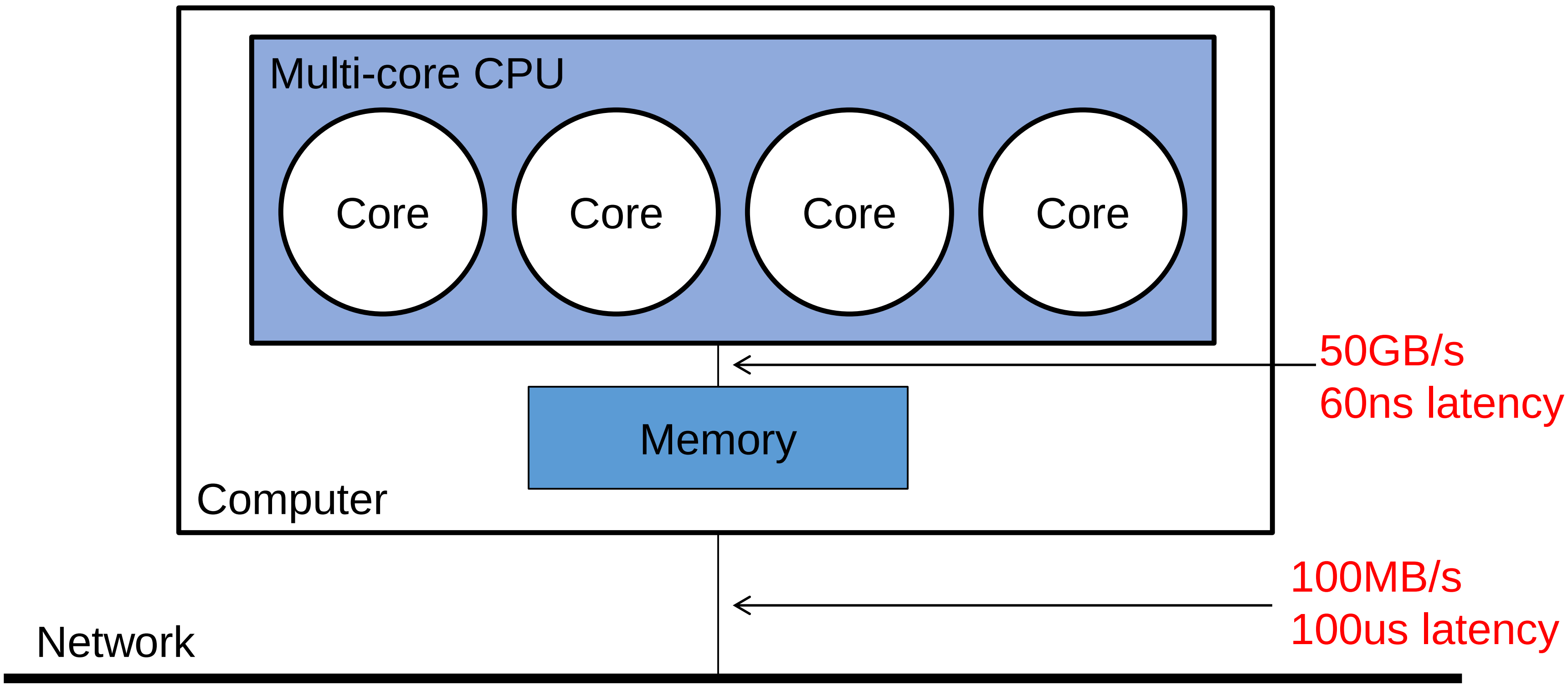
Multi-Core Hardware



Distributed Computing



How Slow Is Communication?



- Network is ~1000x slower than memory
  - Faster on a supercomputer, but still relatively slow

Distributed Algorithms in Gurobi 6.0

- 3 distributed algorithms in version 6.0
  - Distributed tuning
  - Distributed concurrent
    - LP (new in 6.0)
    - MIP
  - Distributed MIP (new in 6.0)

Distributed Tuning

- Tuning:
  - MIP has lots of parameters
  - Tuning performs test runs to find better settings
- Independent solves are obvious candidate for parallelism
- Distributed tuning a clear win
  - 10x faster on 10 machines
- Hard to go back once you have tried it

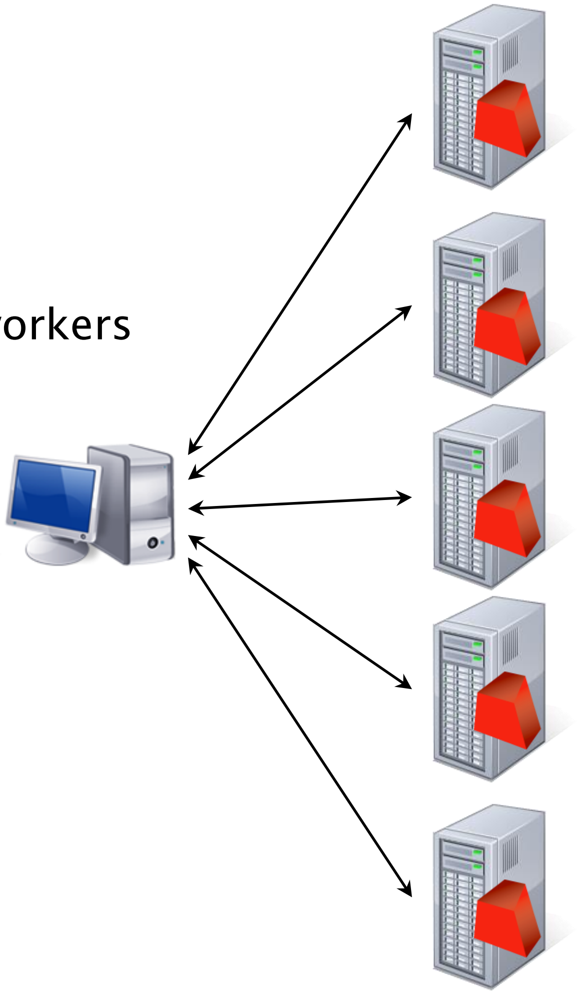
Concurrent Optimization

- Run different algorithms/strategies on different machines/cores
  - First one that finishes wins
- Nearly ideal for distributed optimization
  - Communication:
    - Send model to each machine
    - Winner sends solution back
- Concurrent LP:
  - Different algorithms:
    - Primal simplex/dual simplex/barrier
- Concurrent MIP:
  - Different strategies
  - Default: vary the seed used to break ties
- Easy to customize via concurrent environments

Distributed MIP

Distributed MIP Architecture

- Manager–worker paradigm
- Manager
  - Send model to all workers
  - Track dual bound and worker node counts
  - Rebalance search tree to put useful load on all workers
  - Distribute feasible solutions
- Workers
  - Solve MIP nodes
  - Report status and feasible solutions
- Synchronized deterministically



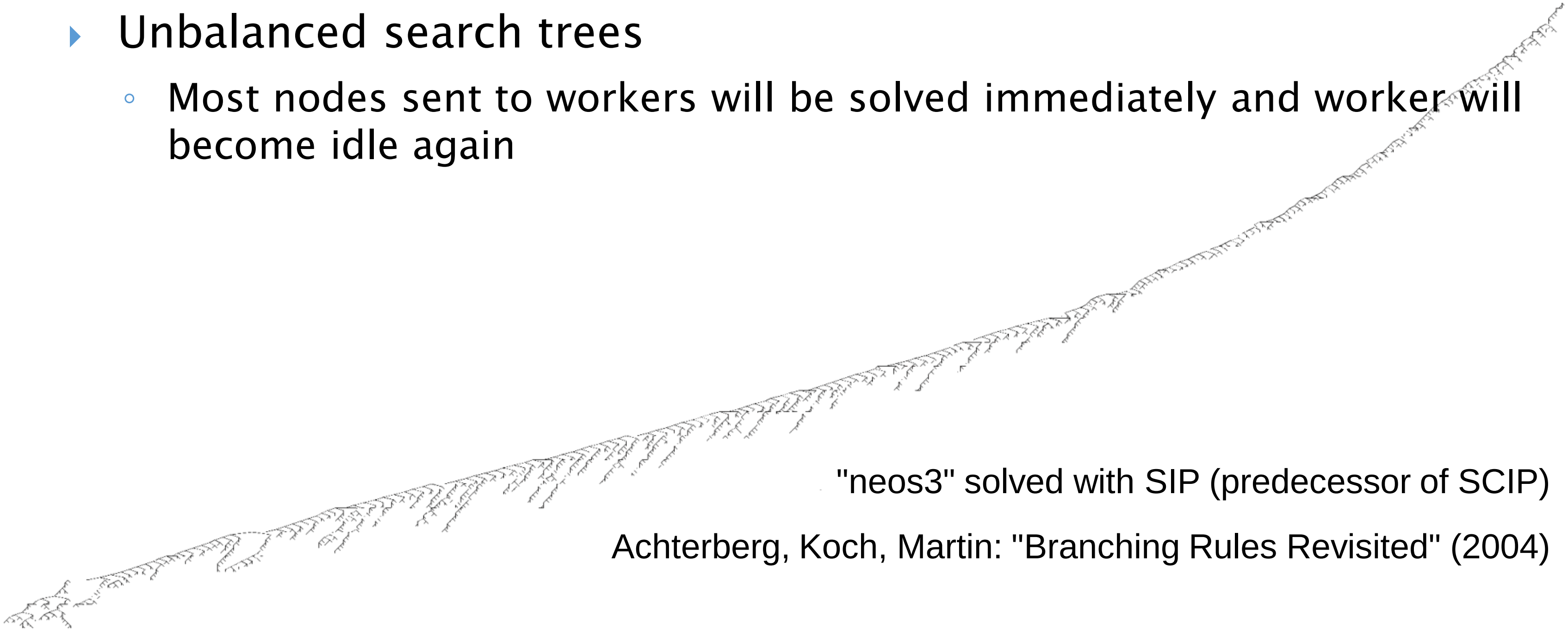
GUROBI OPTIMIZATION

Distributed MIP Phases

- Racing ramp-up phase
  - Distributed concurrent MIP
    - Solve same problem individually on each worker, using different parameter settings
    - Stop when problem is solved or “enough” nodes are explored
    - Choose a “winner” – worker that made the most progress
- Main phase
  - Discard all worker trees except the winner's
  - Collect active nodes from winner, distribute them among now idle workers
  - Periodically synchronize to rebalance load

Bad Cases for Distributed MIP

- Easy problems
  - Why bother with heavy machinery?
- Small search trees
  - Nothing to gain from parallelism
- Unbalanced search trees
  - Most nodes sent to workers will be solved immediately and worker will become idle again

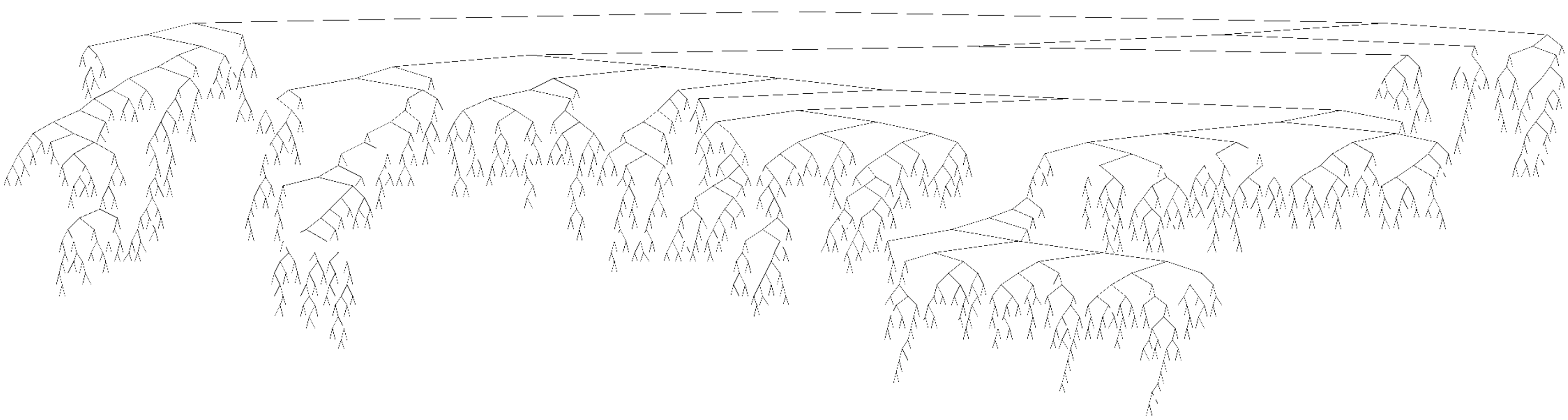


"neos3" solved with SIP (predecessor of SCIP)

Achterberg, Koch, Martin: "Branching Rules Revisited" (2004)

Good Cases for Distributed MIP

- Large search trees
- Well-balanced search trees
  - Many nodes in frontier lead to large sub-trees



"vpm2" solved with SIP (predecessor of SCIP)

Achterberg, Koch, Martin: "Branching Rules Revisited" (2004)

Performance

MIPLIB 2010 Testset

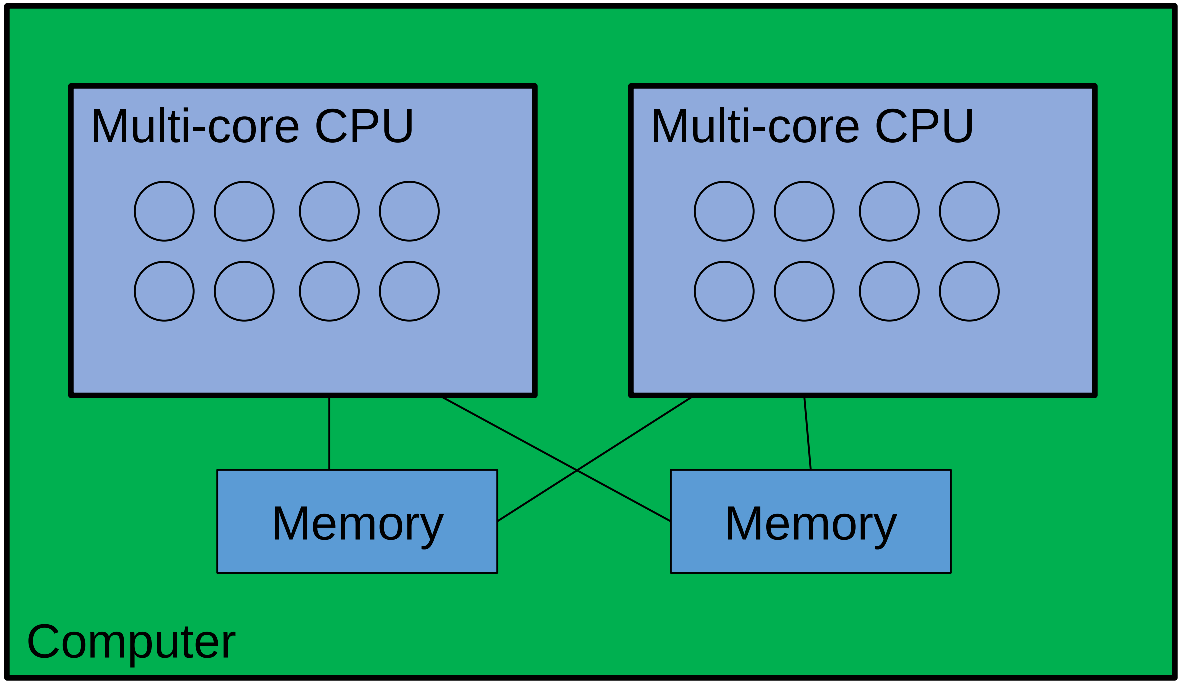
- MIPLIB 2010 test set…
  - Set of 361 mixed-integer programming models
  - Collected by academic/industrial committee
- MIPLIB 2010 benchmark test set…
  - Subset of the full set – 87 of the 361 models
    - Those that were solvable by 2010 codes
    - (Solvable set now includes 206 of the 361 models)
- Notes:
  - Definitely not intended as a high-performance computing test set
    - More than 2/3 solve in less than 100s
    - 8 models solve at the root node
    - ~1/3 solve in fewer than 1000 nodes

Three Views of 16 Cores

- Consider three different tests, all using 16 cores:
  - On a 16-core machine:
    - Run the standard parallel code on all 16 cores
    - Run the distributed code on four 4-core subsets
  - On four 4-way machines:
    - Run the distributed code
- Which gives the best results?

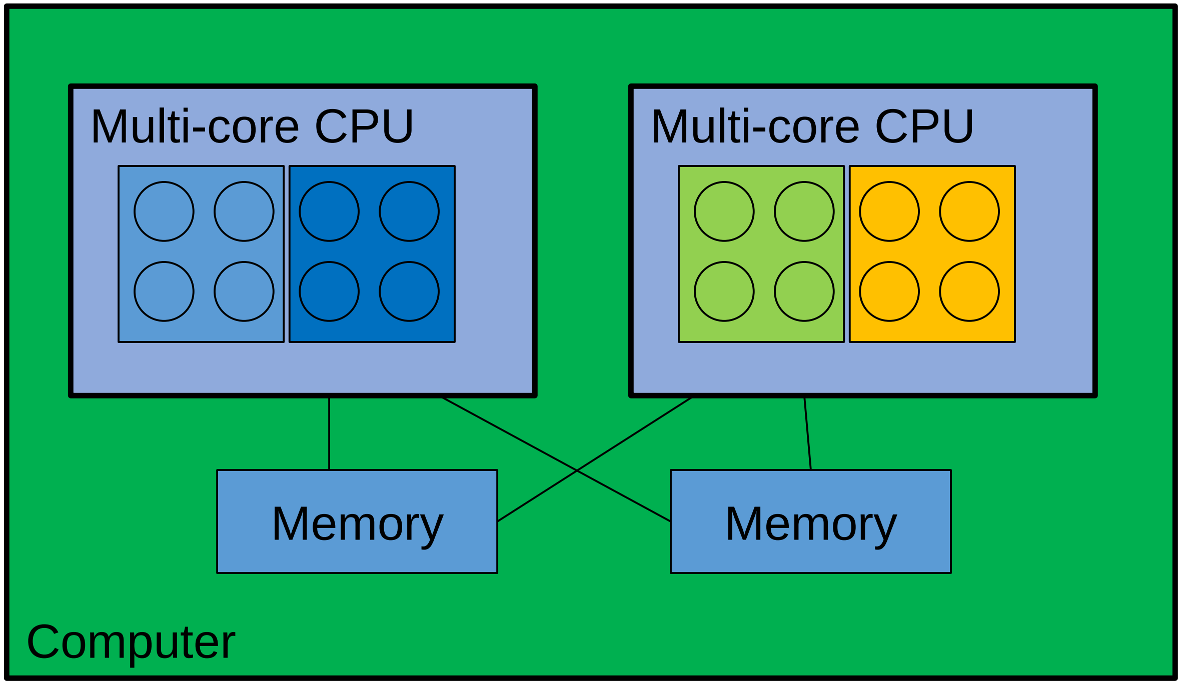
Parallel MIP on 1 Machine

- Use one 16-core machine:



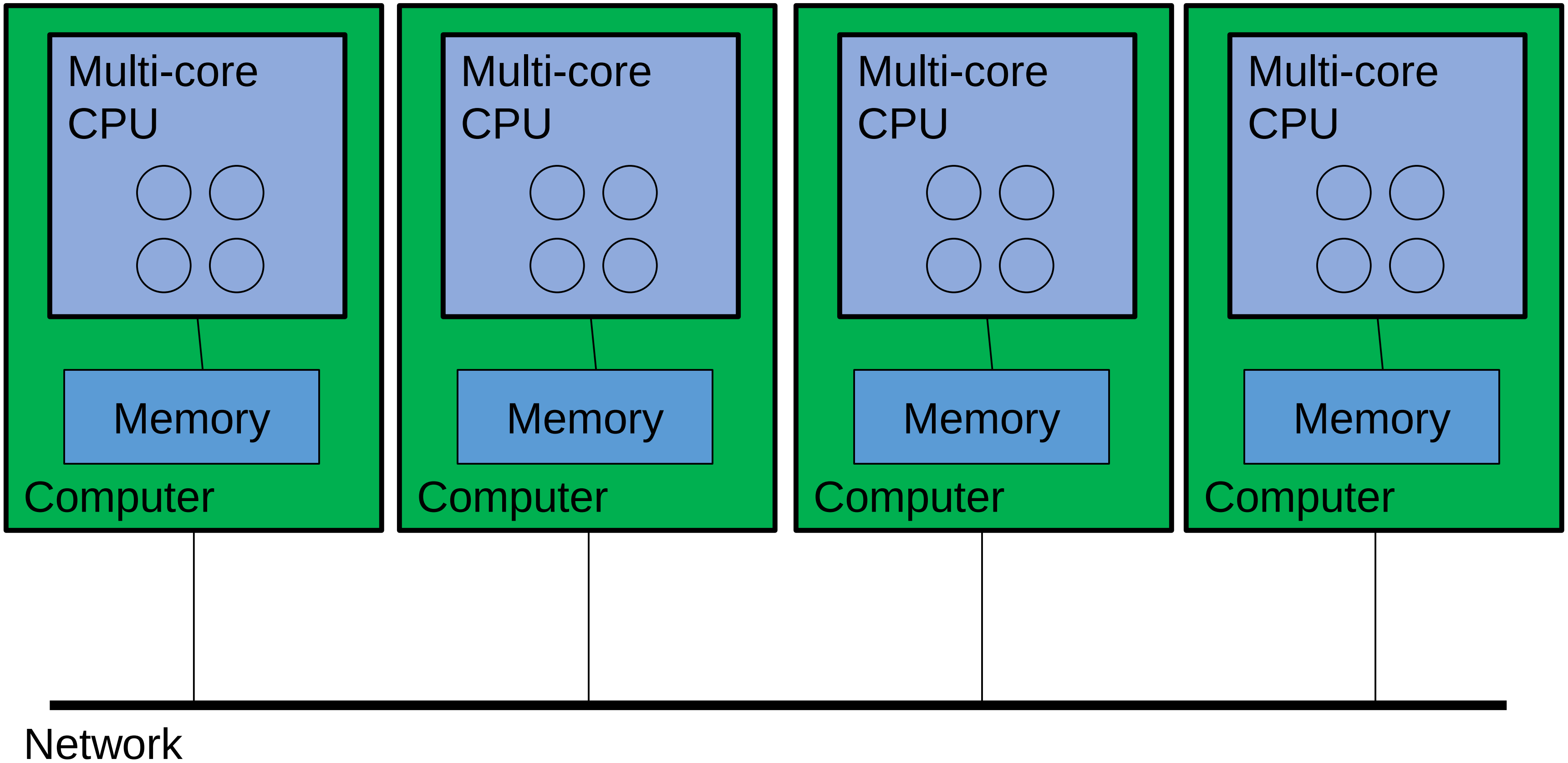
Distributed MIP on 1 machine

- Treat one 16-core machine as four 4-core machines:



Distributed MIP on 4 machines

- Use four 4-core machines



Performance Results

- Using one 16-core machine (MIPLIB 2010, baseline is 4-core run on the same machine)…

Config	>1s	>100s
One 16-core	1.57x	2.00x
Four 4-core	1.26x	1.82x

- Better to run one-machine algorithm on 16 cores than treat the machine as four 4-core machines
  - Degradation isn't large, though

Performance Results

- Comparing one 16-core machine against four 4-core machines (MIPLIB 2010, baseline is single-machine, 4-core run)…

Config	>1s	>100s
One 16-core machine	1.57x	2.00x
Four 4-core machines	1.43x	2.09x

- Given a choice…
  - Comparable mean speedups
  - Other factors…
    - Cost: four 4-core machines are much cheaper
    - Admin: more work to admin 4 machines

Distributed Algorithms in 6.0

- MIPLIB 2010 benchmark set
  - Intel Xeon E3-1240v3 (4-core) CPU
  - Compare against 'standard' code on 1 machine

Machines	>1s			>100s		
	Wins	Losses	Speedup	Wins	Losses	Speedup
2	40	16	1.14x	20	7	1.27x
4	50	17	1.43x	25	2	2.09x
8	53	19	1.53x	25	2	2.87x
16	52	25	1.58x	25	3	3.15x

Some Big Wins

- Model seymour
  - Hard set covering model from MIPLIB 2010
  - 4944 constraints, 1372 (binary) variables, 33K non-zeroes

Machines	Nodes	Time (s)	Speedup
1	476,642	9,267	–
16	1,314,062	1,015	9.1x
32	1,321,048	633	14.6x

Some Big Wins

- Model a1c1s1
  - lot sizing model from MIPLIB 2010
  - 3312 constraints, 3648 variables (192 binary), 10k non-zeros

Machines	Nodes	Time (s)	Speedup
1	3,510,833	17,299	–
49	9,761,505	1,299	13.3x

Distributed Concurrent Versus Distributed MIP

- MIPLIB 2010 benchmark set (versus 1 machine run):
  - >1s

Machines	Concurrent	Distributed
4	1.26x	1.43x
16	1.40x	1.58x

  - >100s

Machines	Concurrent	Distributed
4	1.50x	2.09x
16	2.00x	3.15x

Distributed MIP – Licensing

- Commercial
  - Not included – must purchase the distributed option
  - Ask your sales representative for benchmarks or pricing
- Academic
  - Named-user: not included in licenses from Gurobi website
  - Site license: includes distributed parallel algorithms

Gurobi 6.5

- Currently in beta phase
- Release scheduled for November 2015

Gurobi 6.5 Enhancements

- Variable hints
- API recorder and replay
- UpdateMode parameter
- BarX attribute to query the best barrier iterate
- OPB file format reader
- APIs
  - Gurobi environments in Python interface
  - IIS support in MATLAB
  - R interface extensions
- Licensing
  - Password protection for token servers
  - Single-use licenses without token server
- Distributed
  - WorkerPort parameter
  - Distributed MIP logging
- Packaging
  - Compute Server encryption routines in separate library
  - Separate libc++ and libstdc++ ports on Mac to support clang++
- Performance improvements

Variable Hints

- Provide hints to the solver which variable should take which value
- Guides heuristics and branching

- VarHintVal attribute:
  - Specifies value for variable
- VarHintPri attribute:
  - Specifies level of confidence in this particular variable value

- Comparison to MIP starts:
  - MIP start is used to provide an initial feasible solution to the solver
    - Is evaluated prior to starting the solution process
    - Provides incumbent if feasible
    - Does not influence solution process if it is not feasible
  - Variable Hints guide the search
    - High quality hints should lead to a high quality solution quickly
      - Either through heuristics or through branching
    - Affects the whole solution process

API Recorder

- Setting "Record" parameter to 1 will produce "recording000.grbr" file
  - Tracks all Gurobi API calls
- Use gurobi_cl recording000.grbr to replay file
  - Replay Gurobi execution independently from your own application
- Use cases:
  - Debug performance issues
    - Measures time spent in API calls (e.g., model building) and algorithms (solving)
  - Identify cases where your program leaks Gurobi models or environments
    - Lists number of models and environments that were never freed by your program
  - Relay exact sequence of commands your program issues to Gurobi technical support in case you run into a question or issue that is difficult to reproduce
    - Just send recording file, instead of having to send the whole application

Gurobi Environments in Python Interface

- Default Python environment is not created until it is first used
- Can be released with the new disposeDefaultEnv method
- Particularly useful in iPython Notebook
  - Previously Gurobi would always consume a license token as long as a notebook was open

Performance Improvements in Gurobi 6.5 beta

Problem Class	>1s				>100s			
	#	Wins	Losses	Speedup	#	Wins	Losses	Speedup
LP	376	105	50	1.08x	135	45	31	1.09x
primal	353	96	53	1.04x	146	49	24	1.06x
dual	329	86	50	1.07x	110	36	23	1.12x
barrier	370	92	38	1.09x	111	44	20	1.23x
QCP	68	9	7	1.01x	18	5	2	1.33x
MILP	1741	930	471	1.36x	753	474	188	1.71x
MIQCP	120	76	23	3.57x	48	39	5	13.41x

- Gurobi 6.0 vs. 6.5β: > 1.00x means that Gurobi 6.5β is faster
- QP and MIQP: test sets too small to be meaningful
- QCP results also questionable due to size of test set

GUROBI OPTIMIZATION

Where does the MIP Performance come from?

- Cuts 24.1%
  - Improved MIR aggregation 11.2%
  - Improved node cut selection 5.1%
  - More sophisticated root cut filtering and abort criterion 4.1%
  - More aggressive implied bound cuts 1.2%
  - More aggressive sub-MIP cuts 0.8%
- Presolve 15.6%
  - Improvements in probing 7.0%
  - Fixed sparse presolve 3.8%
  - Merging parallel integer columns with arbitrary scalars 1.4%
  - Disconnected components in presolve 1.3%
  - More stable non-zero cancellation 0.7%
  - Aggregating symmetric continuous variables 0.6%
- Branching 7.7%
  - Replaced 10-5 threshold by 10-8 2.6%
  - Follow-on branching 2.4%
  - Using reduced costs as pseudo-costs 1.3%
  - Modified threshold in implications based tie-breaking 1.2%

Where does the MIP Performance come from?

MIP/LP integration		7.5%
◦ Adjusting pi to get stronger reduced costs	3.8%	
◦ Improvements in simplex pricing	3.6%	
Heuristics		3.5%
◦ New heuristic running in parallel to the root node	2.4%	
◦ Randomization in fix-and-dive heuristics	1.1%	
Node presolve		3.9%
◦ Improved conflict analysis	1.8%	
◦ More node bound strengthening	1.4%	
◦ Slightly faster propagation	0.7%	
Compiler		2.0%
◦ Switched to Intel 2016 compiler	2.0%	

Where does the MIQCP Performance come from?

Change	>1s				>100s			
	#	Wins	Losses	Speedup	#	Wins	Losses	Speedup
cone disaggr.	116	40	0	1.79x	25	16	0	4.43x
branching thr.	106	30	6	1.39x	24	10	6	2.34x
impr. presolve	114	12	2	1.16x	34	5	0	1.40x
impr. OA cuts	110	48	25	1.51x	46	27	7	2.30x

- Cone disaggregation
  - See Vielma, Dunning, Huchette, Lubin (2015)
- Branching threshold
  - See MILP: changing 10^{-5} to 10^{-8}
- Improved presolve
  - Detecting one particular structure to improve bound strengthening
- Improved outer approximation cuts
  - See Günlük and Linderoth (2011)

GUROBI OPTIMIZATION

Projected Roadmap

- Performance
- Parallelism
  - Improve performance on 12+ core systems
  - Improve distributed MIP performance and scalability
- Enhanced Gurobi Cloud offering
- Piecewise-linear extensions
  - PWL objective in MIP
  - PWL constraint coefficients in MIP
- Automatic linearization of common logical constraints

Thank You

- New academic users should visit our Academic Licensing page to get free academic versions of Gurobi
  - http://www.gurobi.com/products/licensing-and-pricing/academic-licensing

- New commercial users should request a free evaluation version of Gurobi either directly from your account rep or by emailing sales@gurobi.com.

- If you have any general questions, please feel free to email info@gurobi.com.