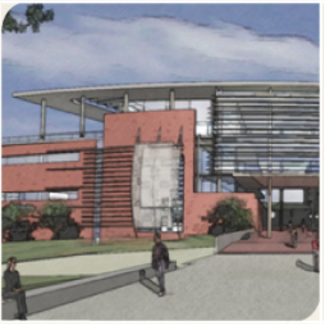
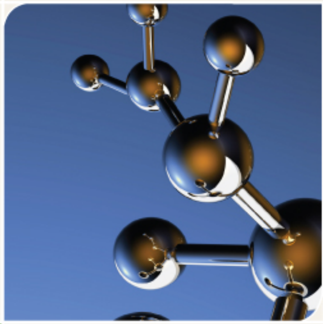
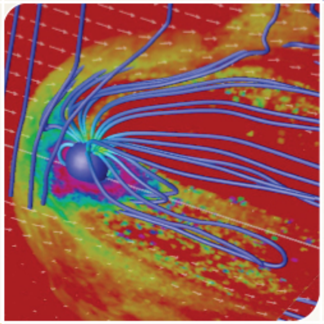
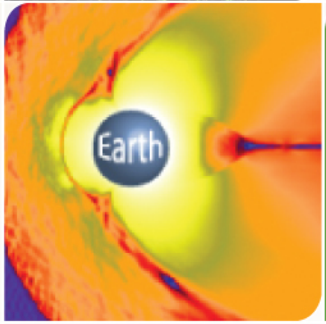
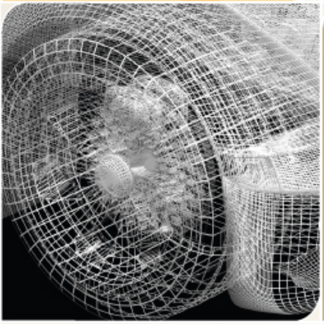


Parallel PIPS-SBB

Multi-level parallelism for 2-stage SMIPS

Lluís-Miquel Munguia, Geoffrey M. Oxberry, Deepak Rajan, Yuji Shinano

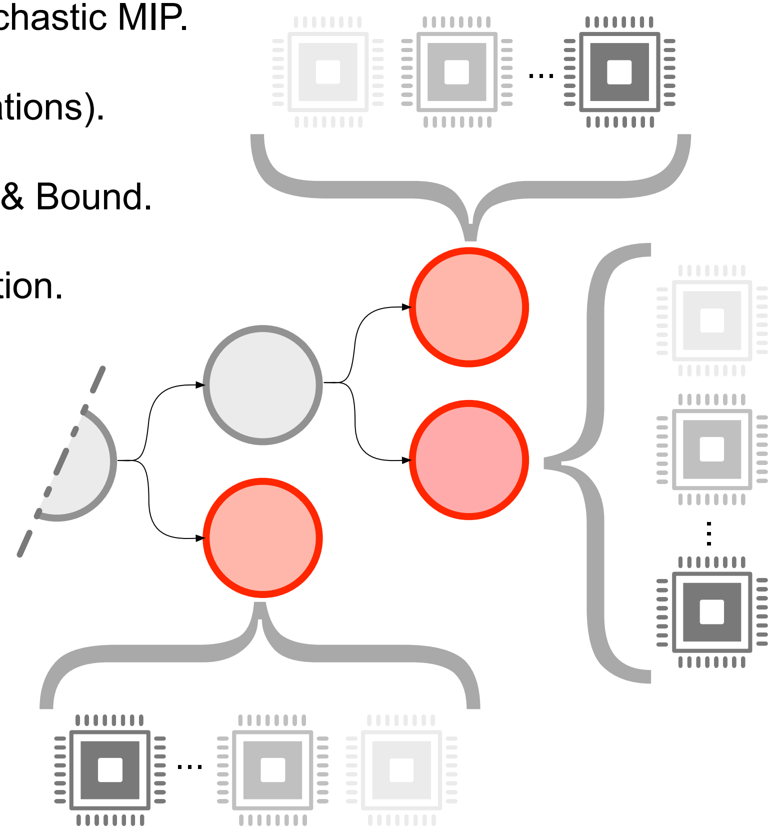


Georgia Tech College of Computing
Computational Science and Engineering

Our contribution

PIPS-PSBB*: Multi-level parallelism for Stochastic Mixed-Integer programs

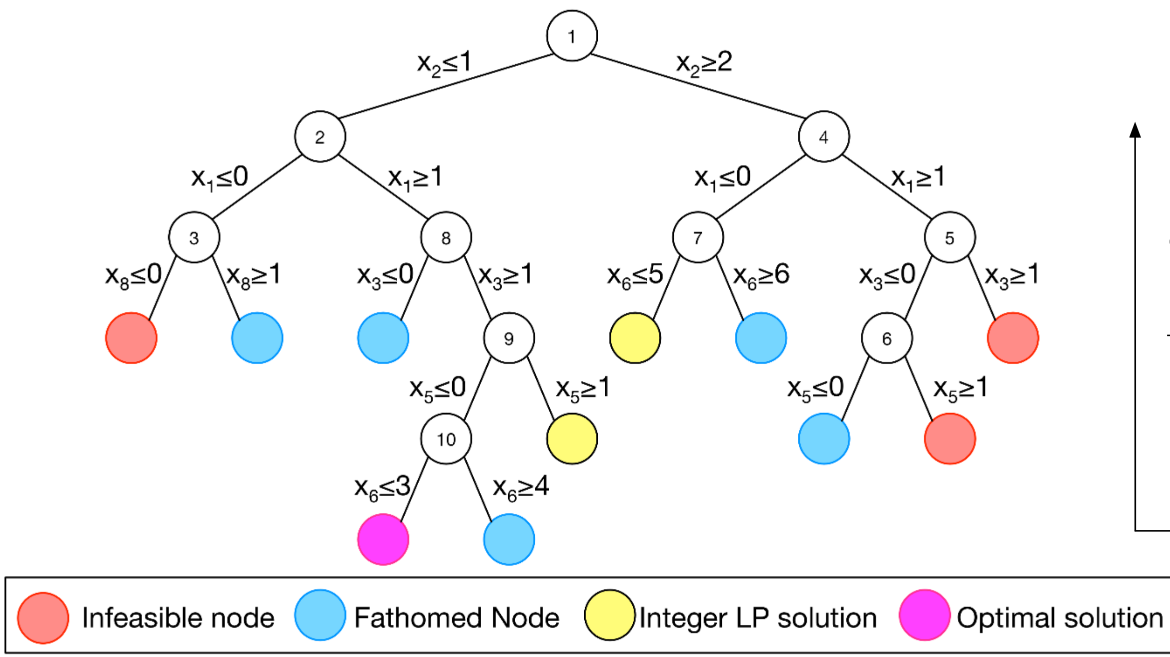
- Fully-featured MIP solver for any generic 2-stage Stochastic MIP.
- Two levels of nested parallelism (B & B and LP relaxations).
- Integral parallelization of every component of Branch & Bound.
- Handle large problems: parallel problem data distribution.
- Distributed-memory parallelization.
- Novel fine-grained load-balancing strategies.
- Actually two(2) parallel solvers:
    - PIPS-PSBB
    - ug[PIPS-SBB,MPI]

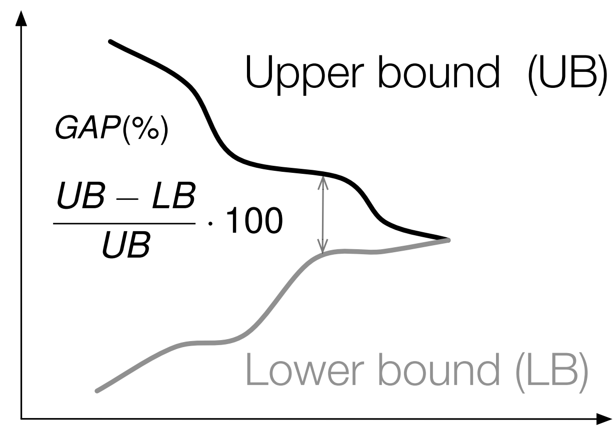


*PIPS-PSBB: Parallel Interior Point Solver – Parallel Simple Branch and Bound

Introduction

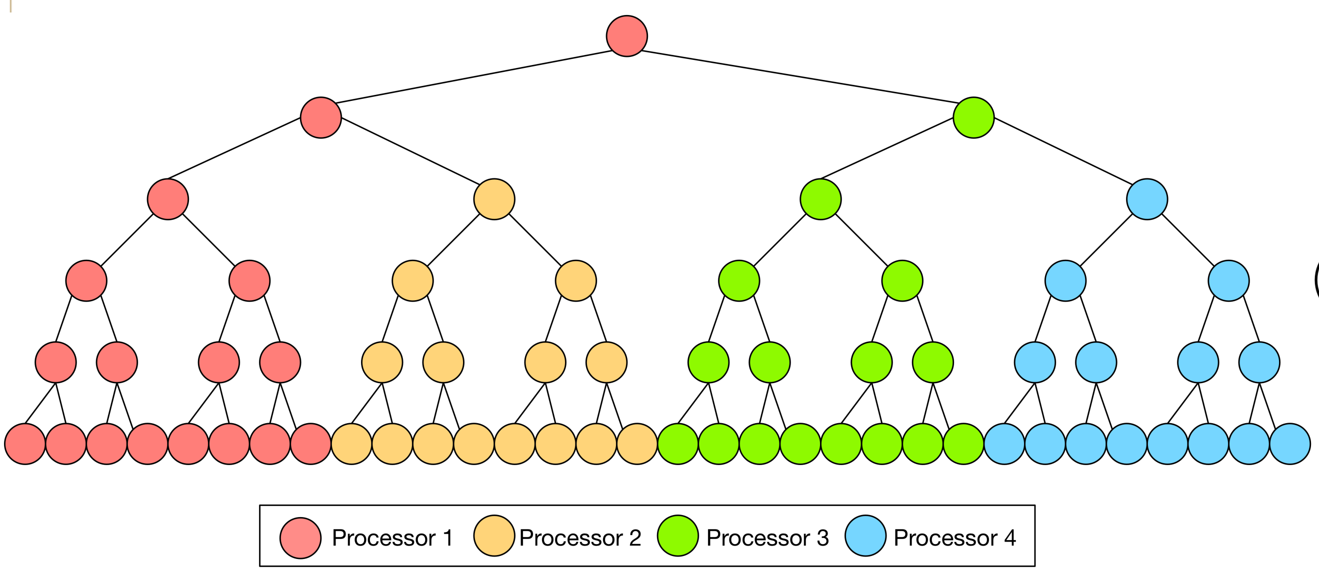
- MIPs are NP-Hard problems: Theoretically and computationally intractable.
- LP-based Branch & Bound allows us to systematically search the solution space by subdividing the problem.
- Upper Bounds (UB) are provided by the integer solutions found along the Branch & Bound exploration. Lower Bounds (LB) are provided by the optimal values of the LP relaxations.

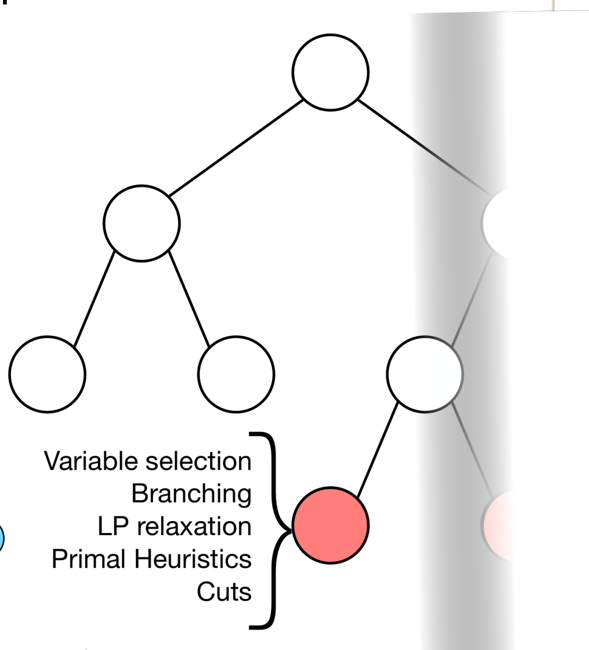




Coarse-grained Parallel Branch and Bound

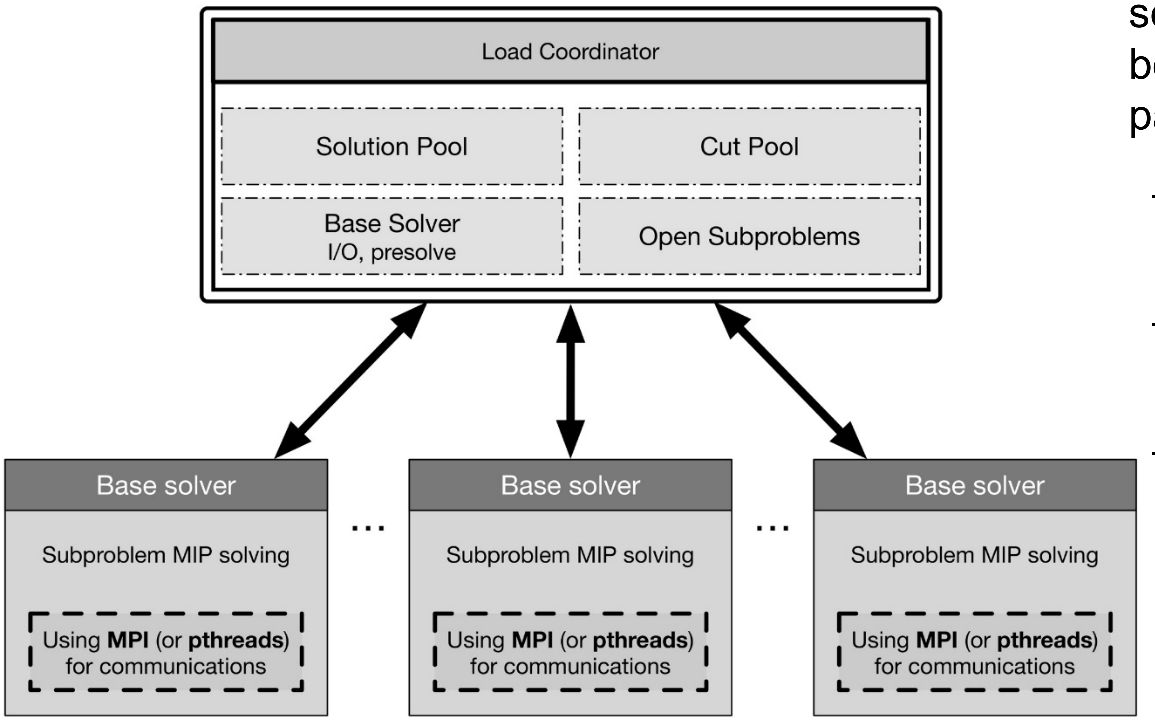
- Branch and bound is straightforward to parallelize: the processing of subproblems is independent.
- Standard parallelization present in most state-of-the-art MIP solvers.
- Processing of a node becomes the sequential computation bottleneck.
- Coarse grained parallelizations are a popular option: Potential performance pitfalls due to a master-slave approach, and relaxations are hard to parallelize.





Coarse-grained Parallel Branch and Bound

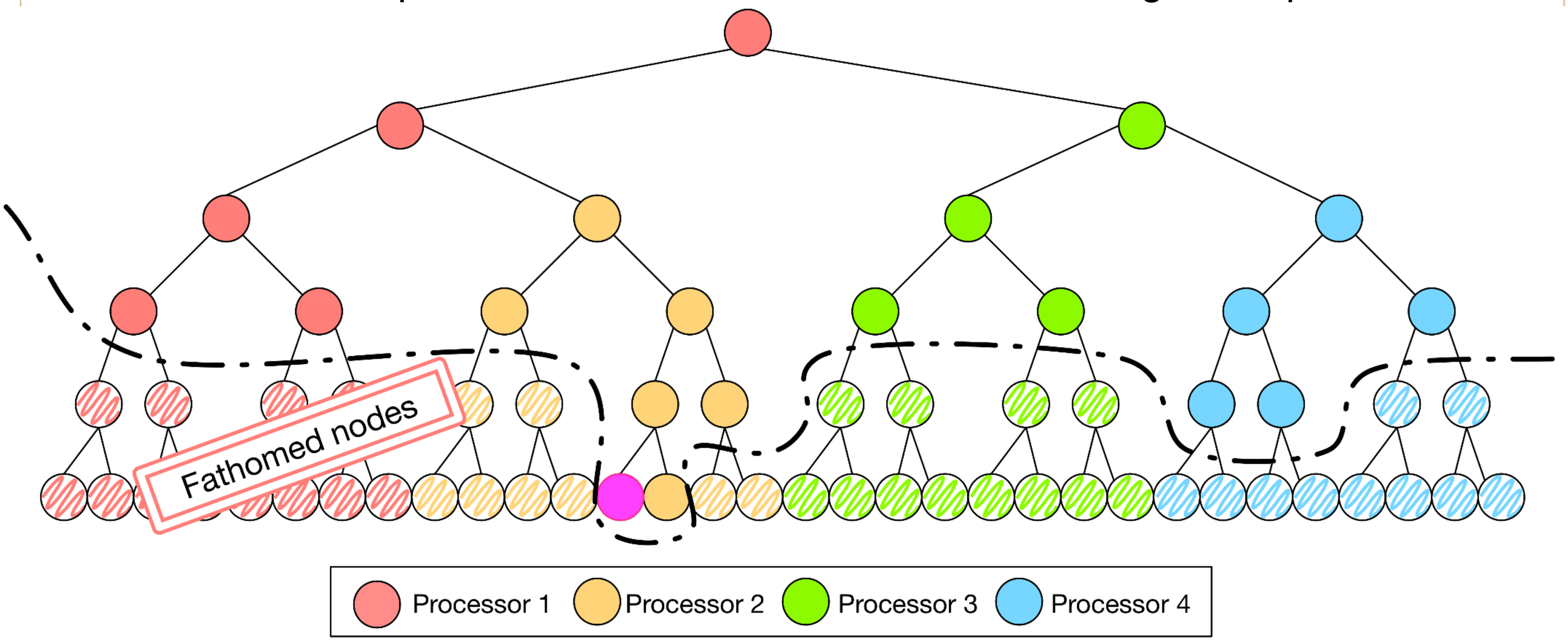
- Branch and Bound exploration is coordinated by a special process or thread.
- Worker threads solve open subproblems using a base MIP solver.



- Centralized communication poses serious challenges: performance bottlenecks and a reduction in parallel efficiency:
  - Communication stress at ramp-up and ramp-down.
  - Limited rebalancing capability: suboptimal distribution of work.
  - Diffusion of information is slow.

Currently available coarse-grained parallelizations

- Coarse-grained parallelizations may scale poorly.
- Extra work is performed when compared to the sequential case.
- Information required to fathom nodes is discovered through the optimization.

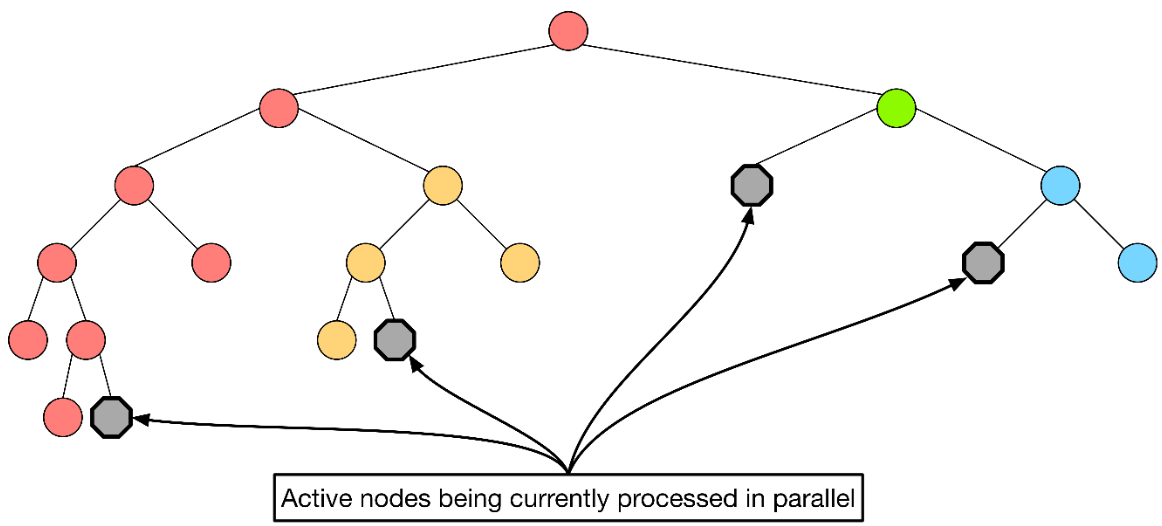


- **Powerful heuristics are necessary to find good feasible solutions early in the search.**

Georgia Tech College of Computing

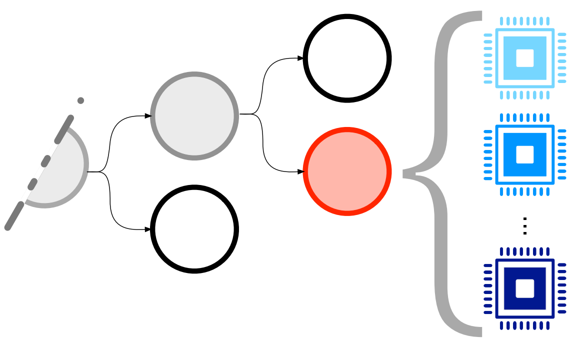
Branch and Bound as a graph problem

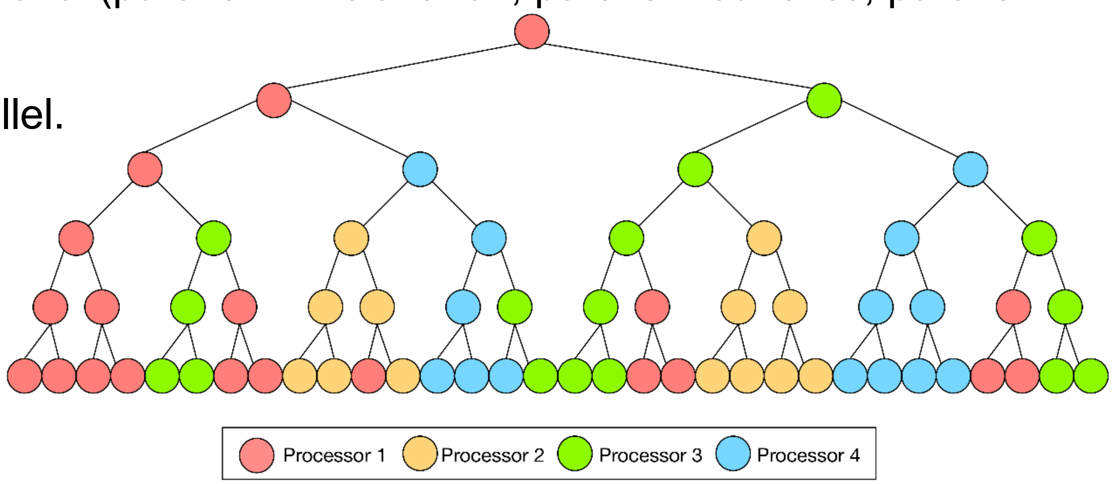
- We can regard parallel Branch and Bound as a parallel graph exploration problem
- Given P processors, we define the frontier of a tree as the set of P subproblems currently being open. The subset currently processed in parallel are the active nodes.
- We additionally define a redundant node as a subproblem, which is fathomable if the optimal solution is known.
- The goal is to increase the efficiency of Parallel Branch and Bound by reducing the number of redundant nodes explored.



Our approach to Parallel Branch and Bound

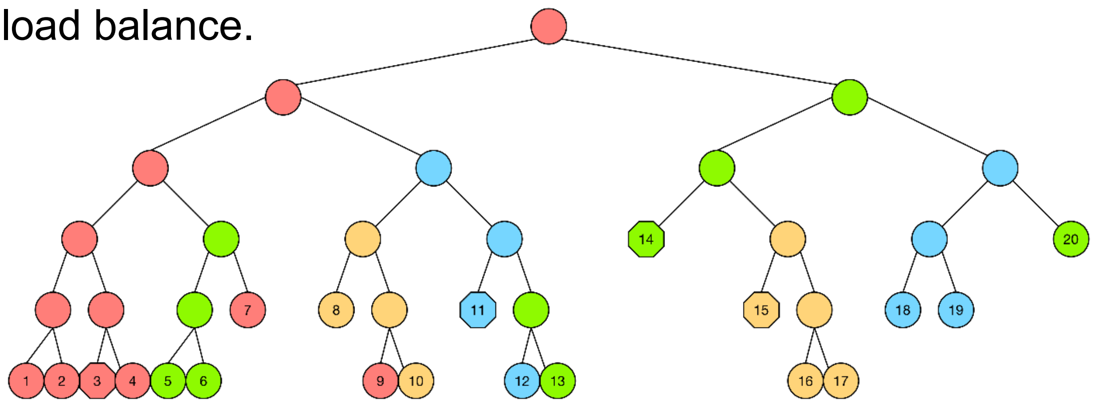
- In order to reduce the amount of redundant nodes explored, the search must fathom subproblems by having high quality primal incumbents and focus on the most promising nodes.
- To increase the parallel efficiency by:
  - Generating a set of active nodes comprised of the most promising nodes.
  - Employing processors to explore the smallest amount of active nodes.
- Two degrees of parallelism:
  - Processing of nodes in parallel (parallel LP relaxation, parallel heuristics, parallel problem branching, …).
  - Branch and Bound in parallel.





Fine-grained Parallel Branch and Bound

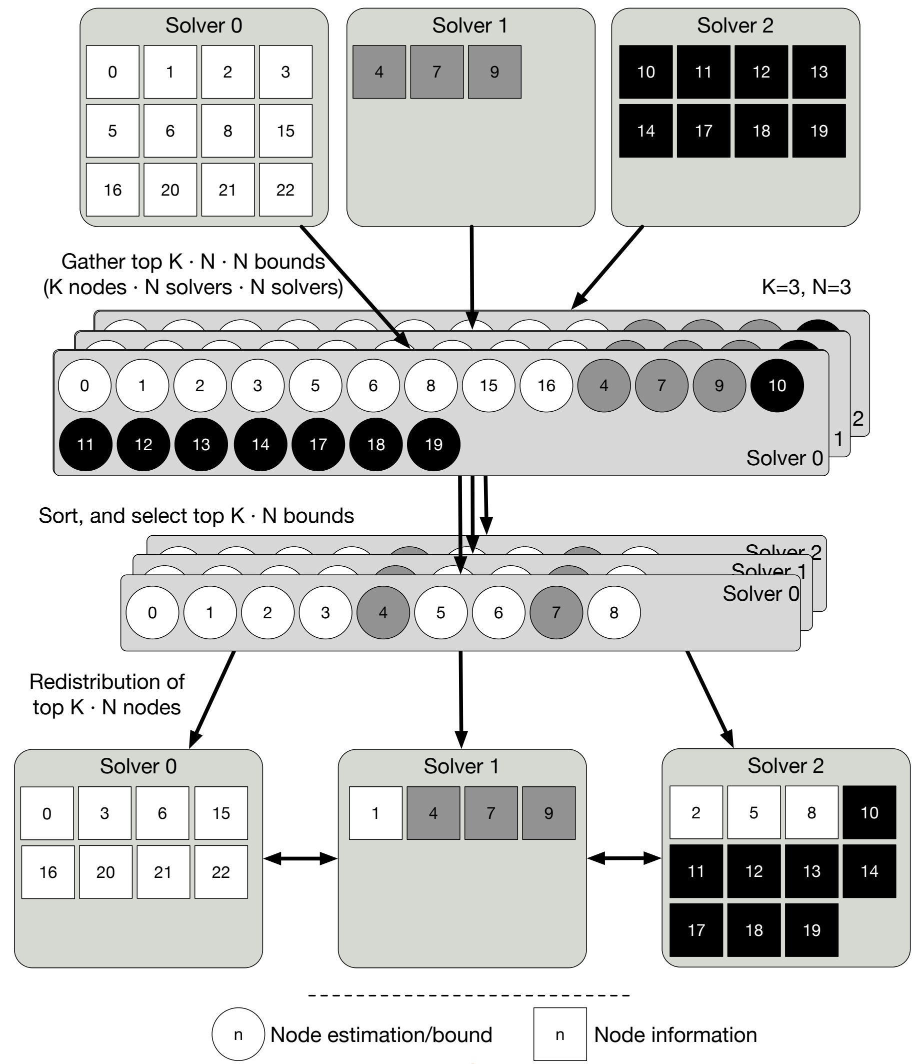
- The smallest transferrable unit of work is a Branch and Bound node.
- Because of the exchange of nodes, queues in processors become a collection of subtrees.
- This allows for great flexibility and a fine-grained control of the parallel effort.
- Coordination of the parallel optimization is decentralized with the objective of maximizing load balance.



	Processor 1	Processor 2	Processor 3	Processor 4
Active nodes	3	15	14	11
Queue	1 2 4 7 9	8 10 16 17	5 6 13 20	12 18 19

All-to-all parallel node exchange

- Load balancing is maintained via synchronous MPI collective communications.
- The lower bound of the most promising K nodes of every processor are exchanged and ranked.
- The top K out of K ·N nodes are selected and redistributed in a round robin fashion.
- Because of the synchronous nature of the approach, communication must be used strategically in order to avoid parallel overheads.
- Node transfers are synchronous, while the statuses of each solver (Upper/lower bounds, tree sizes, times, solutions, …) are exchanged asynchronously.



Georgia Tech College of Computing

Stochastic Mixed Integer Programming: an overview

- Stochastic programming models optimization problems involving uncertainty.

- We consider two-stage stochastic mixed-integer programs (SMIPs) with recourse:
  - 1st stage: deterministic "now" decisions
  - 2nd stage: depends on random event & first stage decisions.

$$\min_x \{c^t x + \mathbb{E}_p[Q(x,\omega)] \mid Ax \le b, x_j \in \mathbb{Z}, \forall j \in I_1\}$$

$$Q(x,\omega) = \min_y \{q^t y \mid Wy \le h - Tx, y_j \in \mathbb{Z}, \forall j \in I_2\}$$

- Cost function includes deterministic variables & expected value function of non-deterministic parameters

Stochastic MIPs and their deterministic equivalent

- We consider deterministic equivalent formulations of 2-stage SMIPs under the sample average approximation
- This assumption yields characteristic dual block-angular structure.

$$\min_x\{c^t x + \mathbb{E}_p[Q(x,\omega)] \mid Ax \le b, x_j \in \mathbb{Z}, \forall j \in I_1\}$$

$$Q(x,\omega) = \min_y\{q^t y \mid Wy \le h - Tx, y_j \in \mathbb{Z}, \forall j \in I_2\}$$

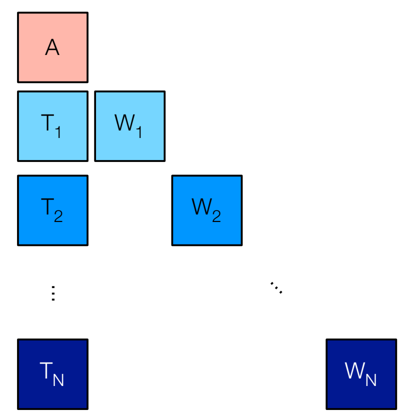
$$\min c^t x$$

s.t.

$$\begin{bmatrix} A & & & & \\ T_1 & W_1 & & & \\ T_2 & & W_2 & & \\ \vdots & & & \ddots & \\ T_N & & & & W_N \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \le \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

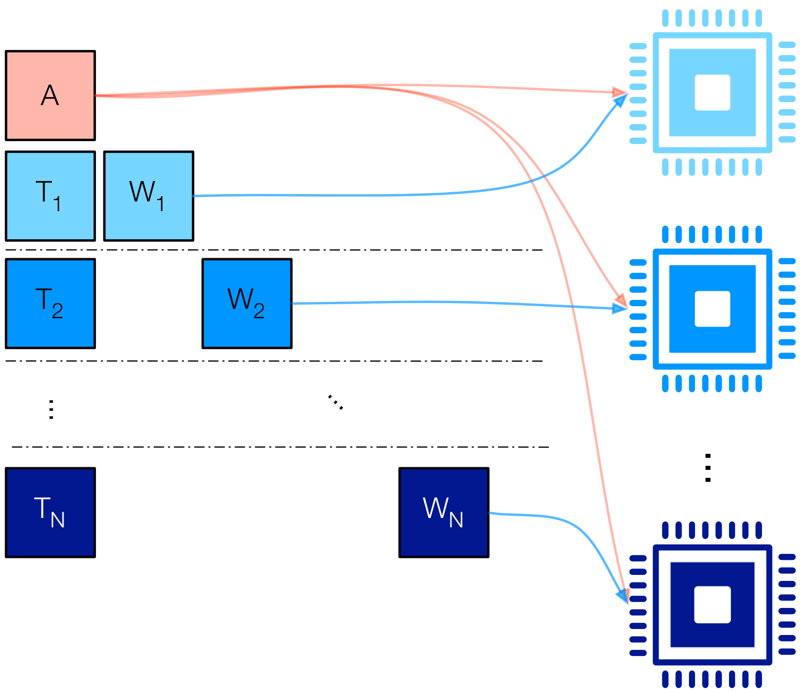
Common constraints

Independent realization scenarios



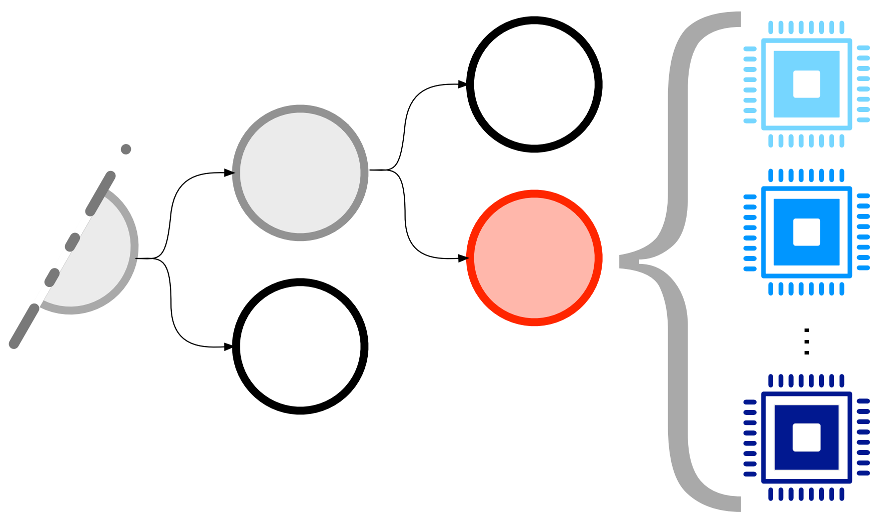
PIPS-PSBB: Design philosophy and features

- PIPS-PSBB is a specialized solver for two-stage Stochastic Mixed Integer Programs that uses Branch and Bound to achieve finite convergence to optimality.
- It addresses each of the the issues associated to Stochastic MIPs:
  - A Distributed Memory approach allows to partition the second stage scenario data among multiple computing nodes.



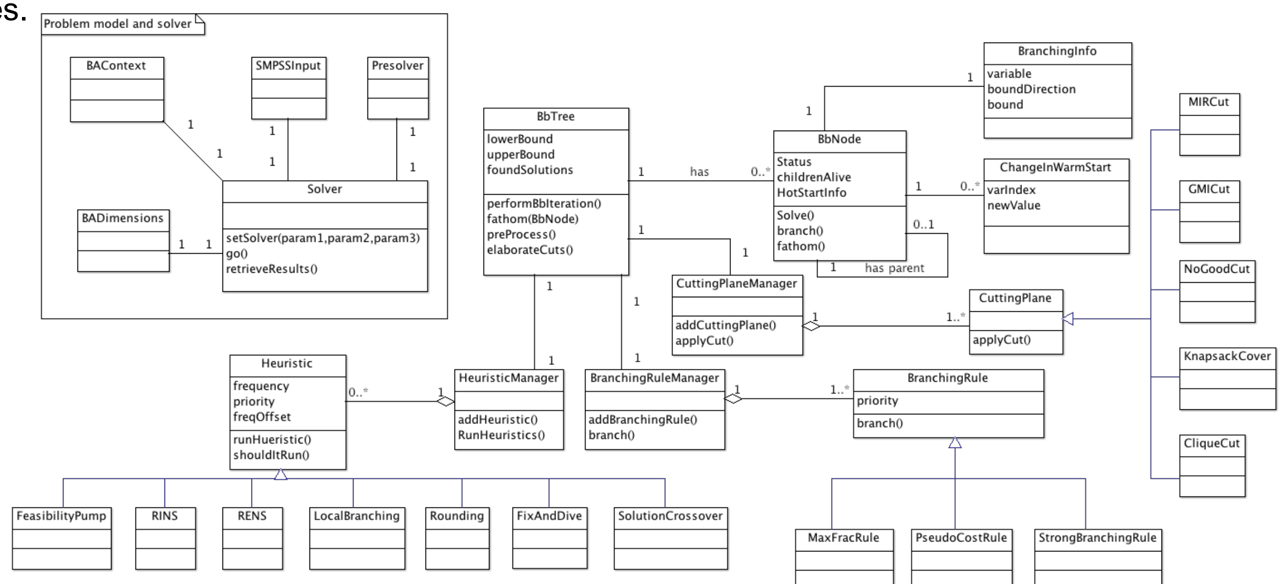
PIPS-SBB: Design philosophy and features

- PIPS-SBB is a specialized solver for two-stage Stochastic Mixed Integer Programs that uses Branch and Bound to achieve finite convergence to optimality.
- It addresses each of the the issues associated to Stochastic MIPs:
  - A Distributed Memory approach allows to partition the second stage scenario data among multiple computing nodes.
  - As the backbone LP solver, we use PIPS-S: a Distributed Memory parallel Simplex solver for Stochastic Linear Programs.



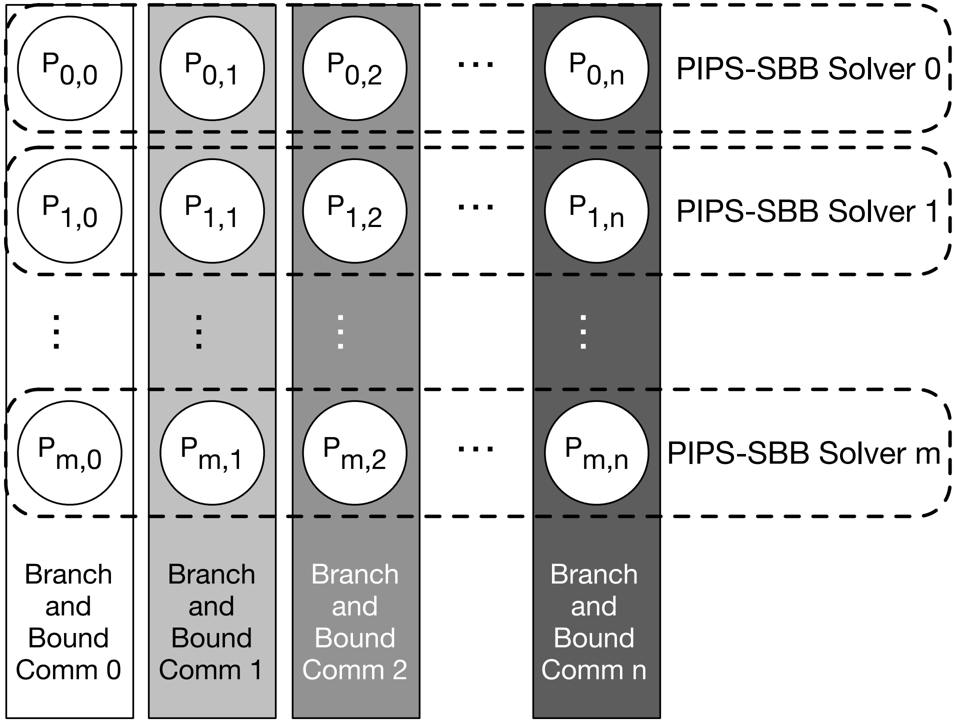
PIPS-SBB: Design philosophy and features

- PIPS-SBB is a specialized solver for two-stage Stochastic Mixed Integer Programs that uses Branch and Bound to achieve finite convergence to optimality.
- It addresses each of the the issues associated to Stochastic MIPs:
  - A Distributed Memory approach allows to partition the second stage scenario data among multiple computing nodes.
  - As the backbone LP solver, we use PIPS-S: a Distributed Memory parallel Simplex solver for Stochastic Linear Programs.
  - PIPS-PSBB has a structured software architecture that is easy to expand in terms of functionality and features.

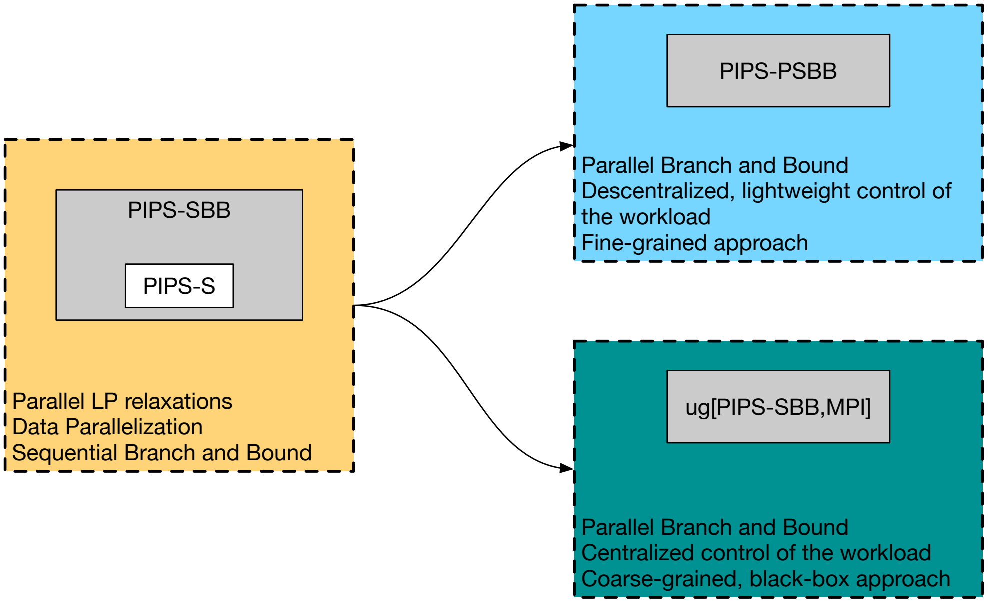


Our approach to Parallel Branch and Bound

- Two levels of parallelism require a layered organization of the MPI processors.
- In the Branch and bound communicator, processors exchange:
  - Branch and Bound Nodes.
  - Solutions.
  - Lower Bound Information.
  - Queue sizes and search status.
- In the PIPS-S communicator, processors perform in parallel:
  - LP relaxations.
  - Primal Heuristics.
  - Branching and candidate selection.
- Strategies for ramp-up:
  - Parallel Strong Branching
  - Standard Branch and Bound
- Strategy for Ramp-down: intensify the frequency of node rebalancing.



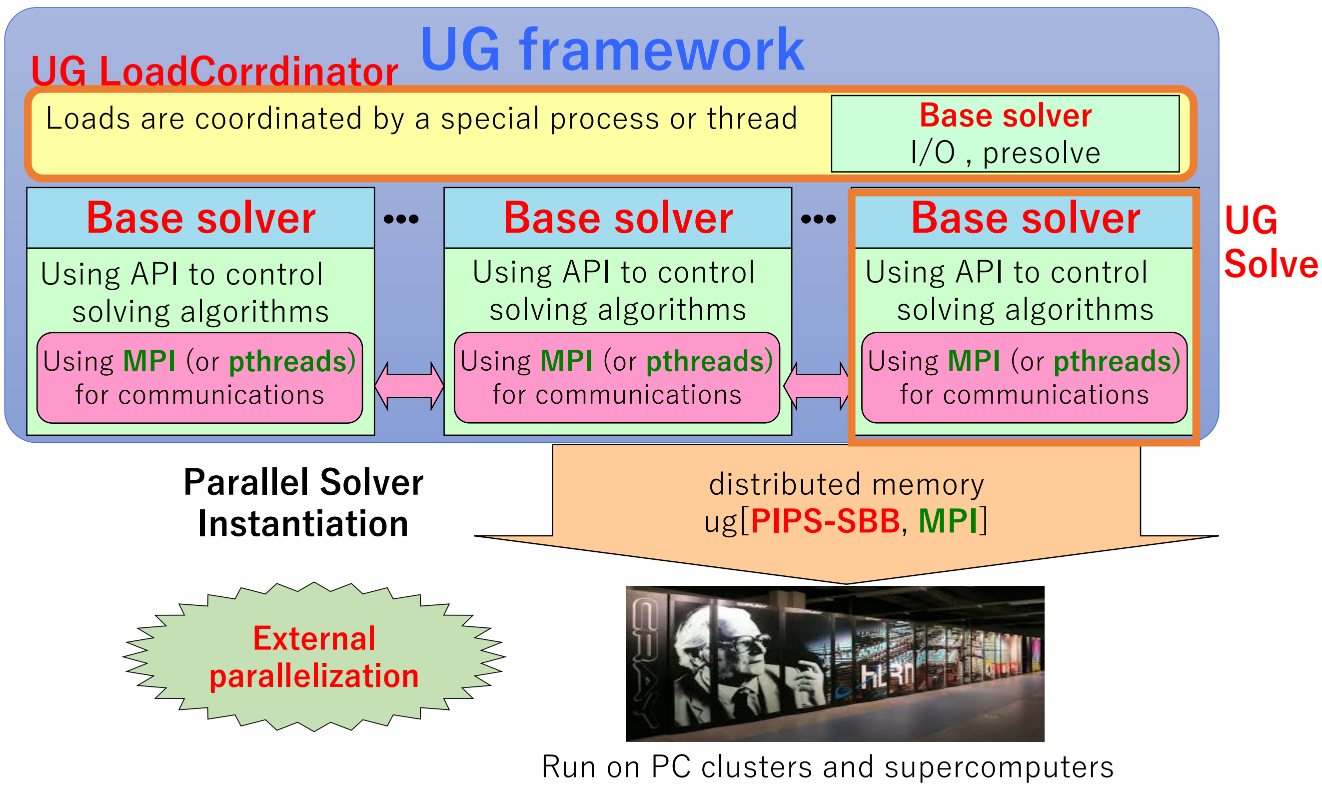
ug[PIPS-SBB,MPI]



- In addition to PIPS-PSBB, we also introduce ug[PIPS-SBB,MPI]: a coarse grained external parallelization of PIPS-SBB.
- UG is a generic framework used to parallelize Branch & Bound based MIP solvers.
  - Exploits powerful performance of state-of-the-art base solvers, such as SCIP, Xpress, Gurobi, and CPLEX.
  - It uses the base solver as a black box.
- UG has been widely applied to parallelize many MIP solvers:
  - Distributed memory via MPI: ug[SCIP,MPI], ug[Xpress,MPI], ug[CPLEX,MPI]
  - Shared-memory via Pthreads: ug[SCIP,Pth], ug[Xpress, Pth]

UG[PIPS-SBB,MPI]

- UG has been successfully used to solve some open MIP problems using more than 80.000 cores. Certainly proven to be scalable.
- ug[PIPS-SBB,MPI] co-developed with Yuji Shinano
- The second MIP solver in the world (after PIPS-PSBB) to use two levels of nested parallelism.



Experimental performance results

- We test our solver on SSLP instances, from the SIPLIB library.
- SSLP instances model server locations under uncertainty.
- Instances are coded as SSLP*m_n_s*, where s represents the number of scenarios.
- Larger number of scenarios means bigger problems
  - LP relaxations of all instances fit in memory, even in CPLEX
  - PIPS-SBB can handle much larger LP relaxations
- Details: see http://www2.isye.gatech.edu/~sahmed/siplib/sslp/sslp.html
- PIPSBB run on the Cab cluster:
  - Each node: Intel Xeon E5-2670, 2.6 GHz, 2 CPUs x 8 cores/CPU
  - 16 cores/node
  - 2 GB RAM/core, 32 GB RAM/node
  - Infiniband QDR interconnect
- CPLEX 12.6.2 used in some comparisons, in Vanilla setting.

Experimental performance results

- We measure parallel performance in terms of speedup, node inefficiency, and communication overhead:
  - Speedup S_p on the time T_p needed to reach optimality by a configuration with p processors with respect to the time needed by a sequential baseline T_1:

$$S_p = \frac{T_1}{T_p}$$

  - Communication overhead: Fraction of time T_{comm} + T_{sync} needed for communication and processor synchronization with respect to the total time of execution T_{exec}:

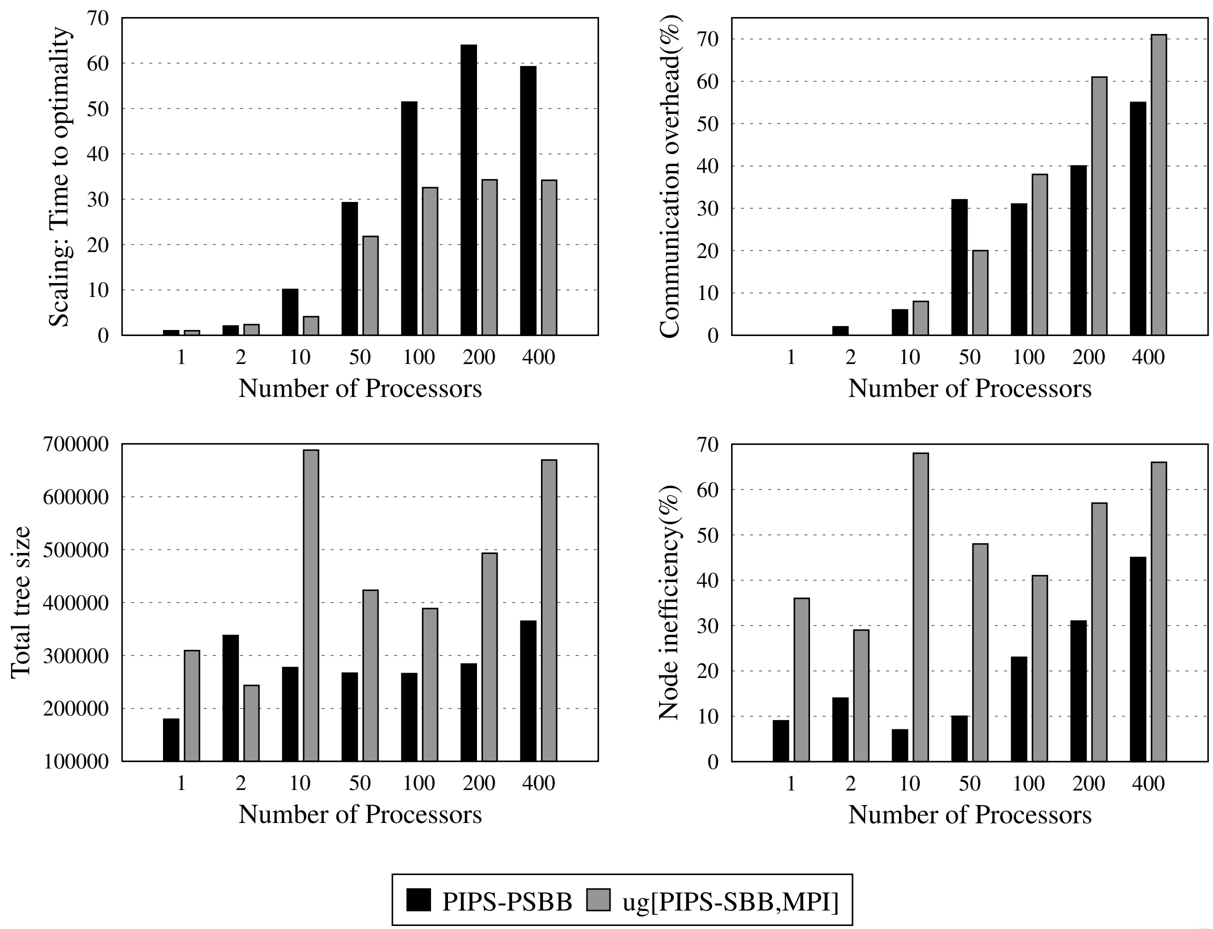
$$C_{ov} = \frac{T_{comm} + T_{sync}}{T_{exec}}$$

  - Node inefficiency: Fraction of redundant nodes explored N_r with respect to the total number of nodes explored N_{total}.

$$N_{ineff} = \frac{N_r}{N_{total}}$$

PIPS-PSBB and ug[PIPS-SBB,MPI]: Performance comparison

Performance comparison between PIPS-PSBB and ug[PIPS-SBB,MPI] when optimizing small instances. sslp_15_45_5 (5 scenarios, 3390 binary variables, 301 constraints)



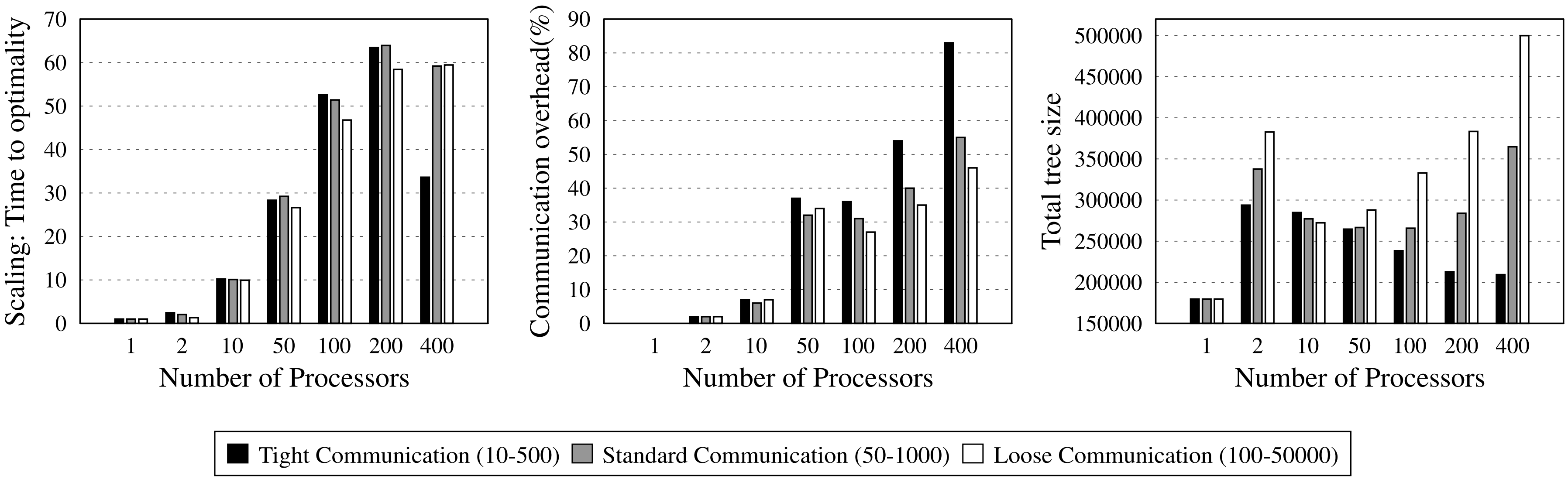
PIPS-PSBB:

- Scales up to 200 cores (66x).
- Total work performed remains within a factor of 2x w.r.t. sequential.
- Communication overhead dominates after 400 cores.
- Node inefficiency grows at a slower rate than ug[PIPS-SBB,MPI].

ug[PIPS-SBB,MPI]:

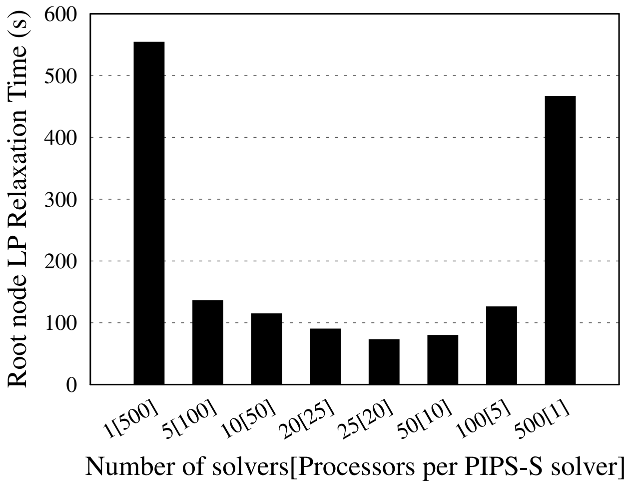
- Scales up to 200 cores (33x).
- Total work varies by processor configuration.
- Higher communication overhead and higher node inefficiency.

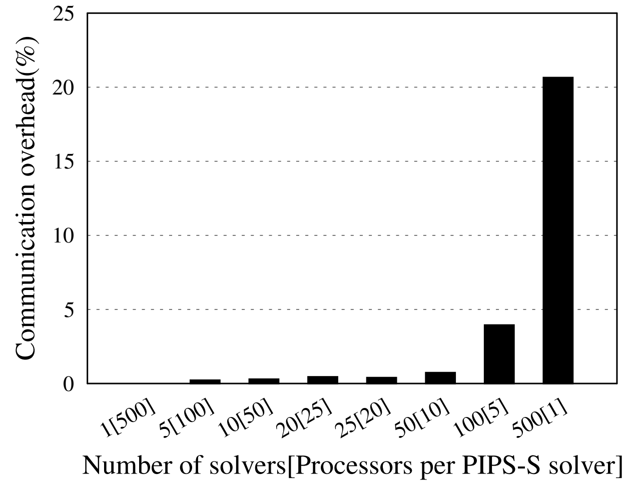
Tuning the communication frequency of PIPS-PSBB

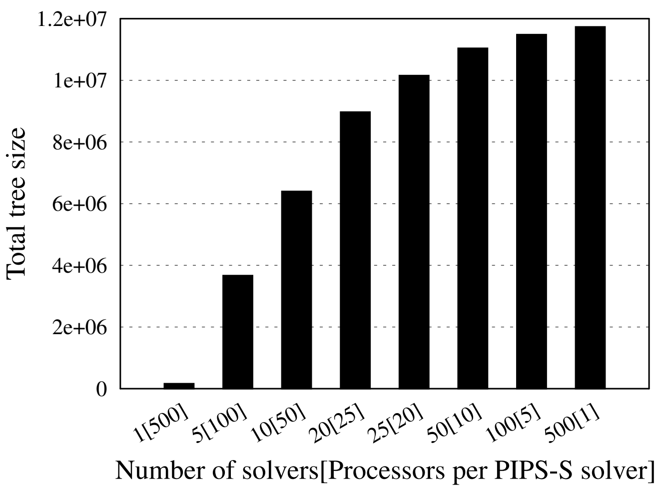


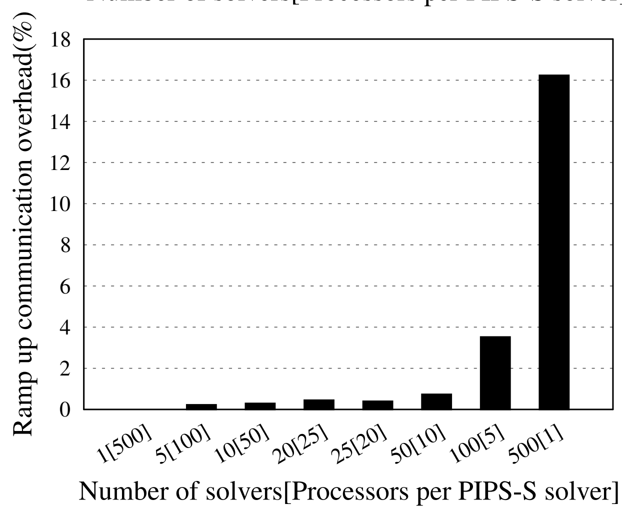
- PIPS-PSBB allows to modify the frequency between synchronous communications.
- Frequency defined with **(x,y)**, where **x** and **y** represent the minimum and maximum number of B&B iterations that must be processed before communication takes place.
- Tighter communication increases communication overheads, but reduces work performed.
- The opposite takes place under loose communication.

PIPS-PSBB Solver performance exposed: sslp_10_50_500 (500 scenarios, 250,010 binary variables, 30,001 constraints)









PIPS-S:

- Speedup to 10 cores is 6x.
- Performance increases up to 20 cores.

PIPS-PSBB:

- Communication overhead minimal except at rampup when LP solver is slow.

PIPS-SBB: Comparison against CPLEX

Performance comparison against CPLEX 12.6.2

Instance	Scenarios	Configuration		PIPS-PSBB	ug[PIPS-SBB,UG]	CPLEX SM		CPLEX DM	
		Solvers	PIPS-S procs	GAP(%) (Time)(s)	GAP(%) (Time)(s)	Procs	GAP(%) (Time)(s)	Procs	GAP(%) (Time)(s)
sslp_5_25_50	50	2	2	(7.45s)	(8.03s)	4	(0.27s)	4	(0.27s)
sslp_5_25_100	100	2	2	(22.37s)	(17.79s)	4	(0.64s)	4	(0.64s)
sslp_15_45_5	5	200	2	(107.11s)	(163.53s)	16	(1.97s)	400	(6.26s)
sslp_15_45_10	10	200	2	0.09%	0.16%	16	(1.81s)	400	(15.04s)
sslp_15_45_15	15	200	2	0.25%	0.30%	16	(7.80s)	400	(15.75s)
sslp_10_50_50	50	200	10	0.13%	0.21%	16	(43.88s)	2000	0.15%(M)
sslp_10_50_100	100	200	10	0.17%	0.20%	16	(221.69s)	2000	0.16%(M)
sslp_10_50_500	500	200	10	0.24%	0.24%	16	4.91%(M)	2000	1.25%(M)
sslp_10_50_1000	1000	200	10	0.24%	0.24%	16	9.91%	2000	6.08%
sslp_10_50_2000	2000	200	10	0.26%	0.26%	16	19.93%	2000	8.11%

Time limit: 1 hour

- Distributed-memory parallelization of CPLEX is often inferior to its shared-memory counterpart.
- Both CPLEX versions run into Memory limits for some problems.

- The superior performance of CPLEX's base solver helps in trivial and small problems.
- PIPS-SBB-based solvers show superior performance for large problems.

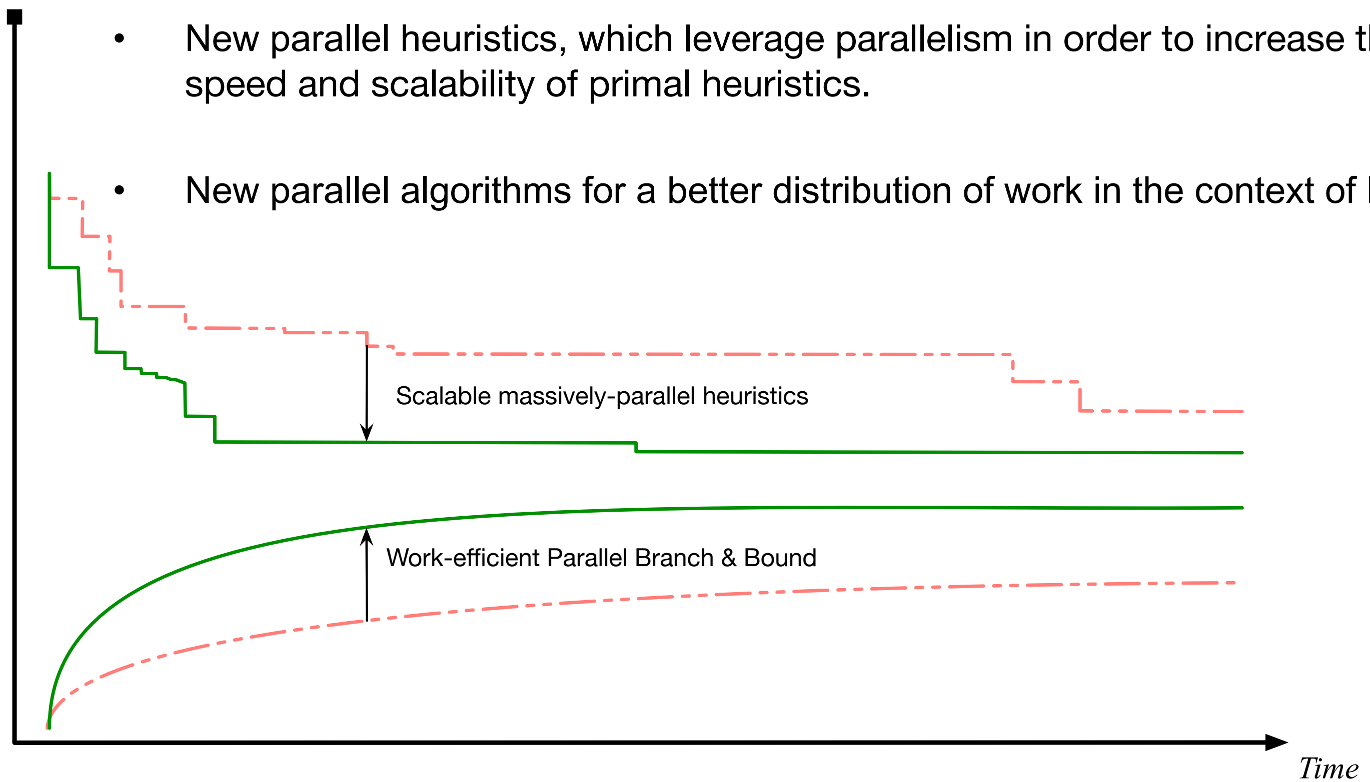
Conclusions

- We developed a light-weight decentralized distributed memory branch-and-bound implementation for PIPS-SBB with two degrees of parallelism:
  - Processing of nodes in parallel (parallel LP relaxation, parallel heuristics, parallel problem branching, …).
  - Branch and Bound in parallel.
- Better parallel efficiency is achieved by focusing the parallel resources in the most promising nodes.
- We try to reduce communication bottlenecks and achieve high processor occupancy via a decentralized control of the tree exploration and a lightweight mechanism for exchanging Branch and Bound nodes.
- Competitive performance to state-of-the-art commercial MIP solvers, in the context of large instances.

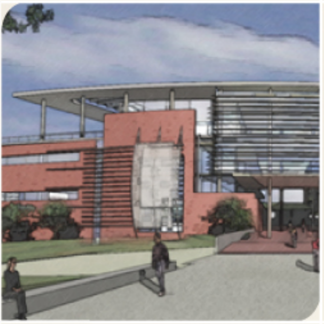
A natural progression in the parallelization of Branch & Bound

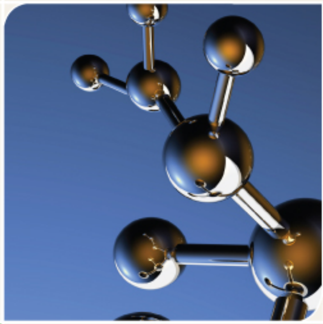
The presented work contributes to the ultimate goal of improving the **parallel efficiency** of Branch & Bound.

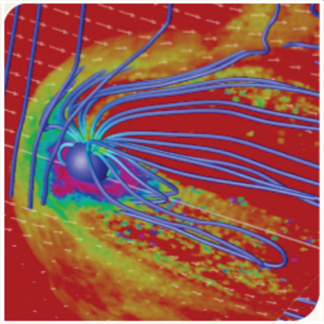
- New parallel heuristics, which leverage parallelism in order to increase the effectiveness, speed and scalability of primal heuristics.
- New parallel algorithms for a better distribution of work in the context of Branch & Bound.

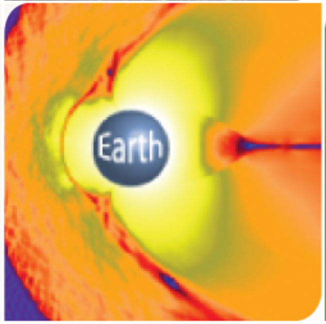


The code of PIPS-PSBB is available at: https://github.com/LLNL/PIPS-SBB

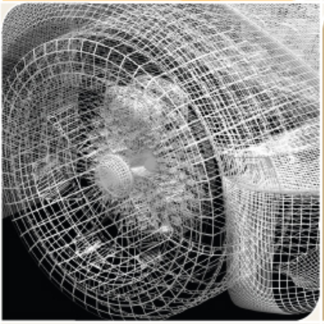








Thank You!



Georgia Tech College of Computing

Computational Science and Engineering