Assessing Performance of Parallel MILP solvers

Ted Ralphs[1]
Stephen J. Maher[2], Yuji Shinano[3]

[1]COR@L Lab, Lehigh University, Bethlehem, PA USA [2]Lancaster University, Lancaster, UK
[3]Zuse Institute Berlin, Berlin, Germany

International Symposium on Mathematical Programming
Bordeaux, France, 5 July 2018

Outline

Outline

Assessing "Effectiveness"

- Fundamental questions we would like to answer.

  - **How to assess algorithm "effectiveness"?**
  - **How to compare solvers with disparate capabilities?**
  - **How to isolate determinants of "effectiveness"?**

- These simple questions are surprisingly difficult to answer!
  - What do we mean by one solver being "better" than another?
  - What is a "fair" way to test?
  - What do we even mean by effectiveness?
- Can we answer these questions by observation without (much) instrumentation?

What Is "Effectiveness"?

- We carefully distinguish between two aspects of what we generally (and loosely) refer to as the "effectiveness" of an algorithm.
- The *computing system* is the combination of a software stack and hardware platform and is a crucial element.
- The *resources* are elements of the system that can be dynamically allocated to a computation (e.g., processors, time).

Performance

- The raw ability of an algorithm to solve an instance or set of instances on a given computing system with *fixed resources*.
- Measures requirement for one resource if others are held constant.

Scalability

- Property of an algorithm deployed in dynamic environment.
- Measures the *tradeoff* between resources (processors and time).

Roadmap for This Talk

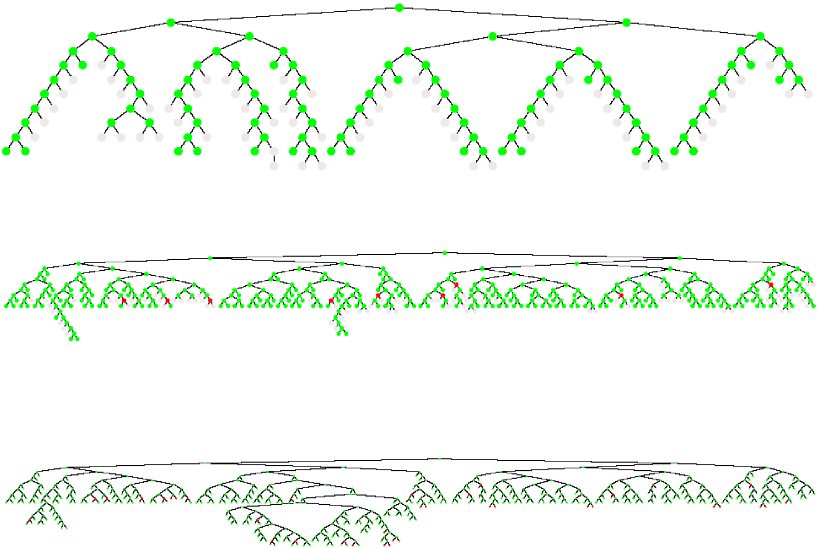
- Focus of this talk will be primarily on assessing scalability.
- Due to time constraints, I'll skip many definitions.
- There is a full paper that goes into much more detail.

Parallelization of Tree Search

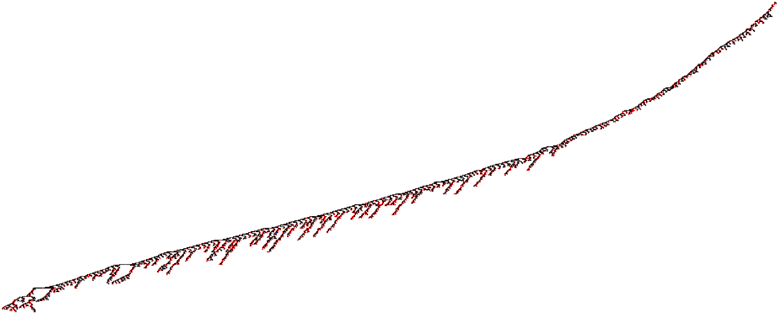
Tree search is easy to parallelize in principle...

- Most straightforwardly, we can parallelize the while loop.
- Naively, this means processing multiple nodes in parallel.
- Applying the successor function turns one task into two!
- This seems to be what is called "embarrassingly parallel"...
- ...but sadly, it's closer to embarrassingly difficult to parallelize!
- We're aiming at a moving target...and with conflicting goals.

Nice Trees



Ugly Trees



Barriers to Scalability: Sophistication of Solvers

- A vast amount of effort has gone into improving the performance of sequential solvers over the past several decades.
- It's been estimated that overall solver performance has improved by a factor of approximately 2 trillion in past decades.
- Unfortunately, major advances in solver technology have mostly made achieving parallel performance *more difficult*.
  - **Solvers are increasingly tightly integrated**.
  - **Work done at the root node is difficult to parallelize**.
  - **Algorithmic focus is on reducing the amount of enumeration**.
  - **Solvers exploit a lot of useful "global" knowledge**.

Branch and cut is not nearly as parallelizable as it seems!

Example: The Knapsack Problem

- We consider the binary knapsack problem:

$$\max\{\sum_{i=1}^{m} p_i x_i : \sum_{i=1}^{m} s_i x_i \le c, x_i \in \{0,1\}, i = 1, 2, \ldots, m\}, \qquad (1)$$

- We implemented a naive LP-based branch-and-bound in the Abstract Library for Parallel Search (ALPS).

P	Node	Ramp-up	Idle	Ramp-down	Wallclock	Eff
4	193057493	0.28%	0.02%	0.01%	586.90	1.00
8	192831731	0.58%	0.08%	0.09%	245.42	1.20
16	192255612	1.20%	0.26%	0.37%	113.43	1.29
32	191967386	2.34%	0.71%	1.47%	56.39	1.30
64	190343944	4.37%	2.27%	5.49%	30.44	1.21

- Perfect scalability! But terrible performance...

...On the Other Hand

- CPLEX output for solving one of these instances...

```
Root node processing (before b&c):
  Real time                 =    0.01 sec. (0.76 ticks)
Sequential b&c:
  Real time                 =    0.00 sec. (0.00 ticks)
                              ------------
Total (root+branch&cut) =    0.01 sec. (0.76 ticks)

Root node processing (before b&c):
  Real time                 =    0.03 sec. (0.74 ticks)
Parallel b&c, 16 threads:
  Real time                 =    0.00 sec. (0.00 ticks)
  Sync time (average)       =    0.00 sec.
  Wait time (average)       =    0.00 sec.
                              ------------
Total (root+branch&cut) =    0.03 sec. (0.74 ticks)
```

- Parallel slowdown! But great performance...

Outline

Outline

Measures of Performance

Single-instance measures

- Time to proven optimality
- Number of nodes to proven optimality
- Time to first feasible solution
- Time to fixed gap
- Gap or primal bound after a time limit
- Primal dual integral (PDI)

Summary Measures

- Mean
- Shifted geometric mean
- Performance profile
- Performance plots
- Histograms

Primal Dual Integral [Berthold, 2013]

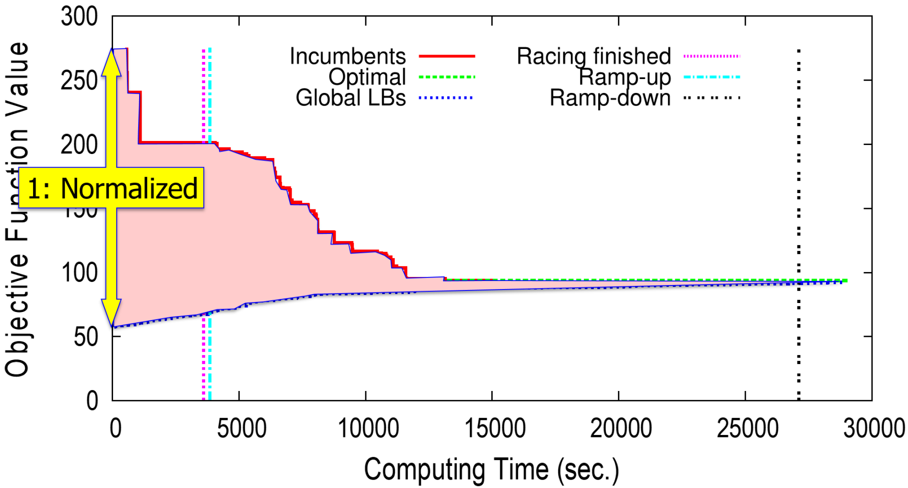


Figure: Example of a PDI plot

The Extended Primal Dual Integral

- The PDI can be inaccurate or nonexistent.
  - No primal (or dual) bounds are known.
  - Primal and dual bounds have different signs.
  - Optimal value is zero.
- Berthold [2013] proposed setting gap to 1 in the first two situations.
  - Note that if the gap stays constant throughout the computation, then it will be normalized to 1 and the PDI will equal the computing time.
  - The method of Berthold [2013] is the same as assuming that the gap is constant through the first part of the computation.
  - This can significantly distort the PDI in some cases.
- The *extended PDI* is our proposal for partially overcoming these issues.

Outline

Parallel Overhead

- The amount of *parallel overhead* determines the scalability.
- "Knowledge sharing" is the main driver of efficiency.

Major Components of Parallel Overhead in Tree Search

- Communication Overhead (cost of sharing knowledge)
- Idle Time
  - Handshaking/Synchronization (cost of sharing knowledge)
  - Task Starvation (cost of *not* sharing knowledge)
  - Memory Contention
  - Ramp Up Time
  - Ramp Down Time
- Performance of Redundant Work (cost of *not* sharing knowledge)

- This breakdown highlights the tradeoff between centralized and decentralized knowledge storage and decision-making.

Parallel Efficiency

Terms

- Sequential runtime: T_s
- Parallel runtime: T_p
- Parallel overhead: $T_o = NT_p - T_s$
- Speedup: $S = T_s/T_p$
- Efficiency: $E = S/N$

- Standard analysis considers change in efficiency on a fixed test set as number of cores is increased.
- *Isoefficiency analysis* considers the increase in problem size to maintain a fixed efficiency as number of cores is increased.

Outline

Outline

Challenges in Analyzing Scalability

- It's exceedingly difficult to construct a test set.
  - Problems need to be solvable by all solvers on single core.
  - Single-core running times should be "long, but not too long"
  - Scalability depends on many factors besides the algorithm itself, including inherent properties of the instances.
  - Different instances scale differently on different solvers.
- It's not clear what the baseline should be.
  - The best known sequential algorithm,
  - The parallel algorithm running on a single core,
  - Or...?
- It's difficult to compare solvers with widely varying performance.
- Results are highly dependent on details of the parallel system.
- Variability in execution!
  - Many sources of variability are difficult to control for.
  - Lack of determinism requires extensive testing.
- Scalability numbers alone don't typically give much insight!

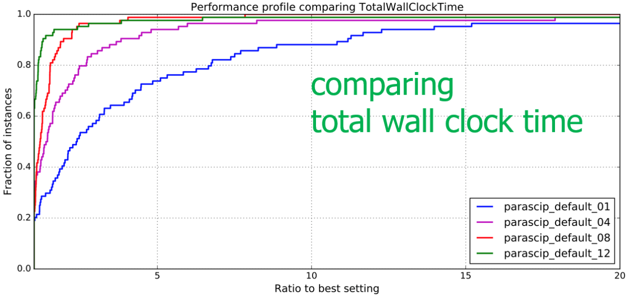
Outline

Alternatives to Classical Scalability Analysis

- Direct measures of overhead.
  - Node throughput
  - Ramp-up/Ramp-down time
  - Idle time/Lock time/Wait time
  - Number of nodes
- Performance, scalability, and cumulative profiles.
- Analysis based on measures of progress.
  - Gap
  - Extended PDI

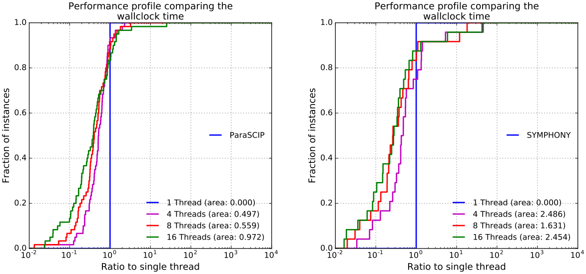
Performance Profiles for Scalability Analysis

- Performance profiles are typically used to compare different algorithms
- They can, however, be used to compare the same algorithm under different conditions.
- For scalability, we compare with differing numbers of threads.
- A down side is that performance profiles compare to virtual best, whereas scalability compares to single-thread.



Scalability Profiles

- Straight performance profile considers ratios against virtual best.
- An alternative is to consider ratios against single thread.
- In the latter case, we must allow ratios less than one.



(a) ParaSCIP (b) SYMPHONY

Figure: Scalability profile of wallclock running time.

Progress-based Analysis

- Traditional scalability analysis asks how much time it takes to do a fixed computation.

Two simple alternatives

- How much computation can be done in a fixed amount of real time but with varying numbers of processors?
- How much computation can be done with fixed compute time but with varying amounts of real time?

- Allowing partial completion of a fixed computation eliminates many of the problems with finding a test set and comparing solvers.
- Both these alternatives depends on having some reliable “measure of progress,” however.
- It is not enough to just measure the “amount of computation”—this is equivalent to measuring utilization and ignoring other overhead.

Measures of Progress

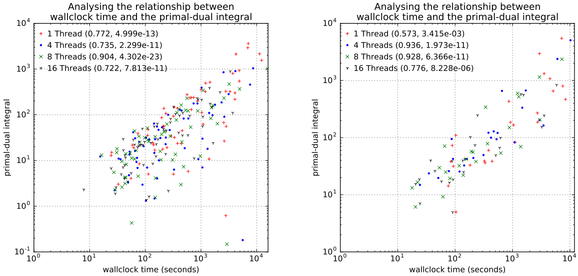
- A *measure of progress* is an estimate of what fraction of a computation has been completed.
- It may be very difficult to predict how much time remains in a computation.
- However, for computations that have already been performed once, it may be possible.
- Measures of progress can be used to assess the effectiveness of algorithms *even if the computation doesn't complete* $\Leftarrow$ Important!
- Possible measures for MILP
  - Gap
  - *Extended PDI*

Gap versus Extended PDI

- Gap
  - Final value is always zero
  - Progress can be “irregular”.
  - Current value doesn’t really indicate now “close” the computation is to finishing.
- Extended PDI
  - Final value can be anything from 0 to the time required for computation (normalized version).
  - Can be normalized to $[0, 1]$, but the final value is still variable.
  - Progress can be “irregular”.
  - Still, it seems to be a reasonable proxy for wallclock running time.

Extended PDI versus Wallclock

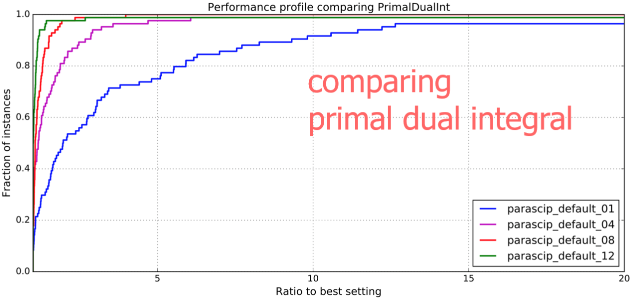
- The below figures show the relationship between wallclock running time and extended PDI for different numbers of threads.
- In general, there is a strong correlation between wallclock and PDI, which is perhaps not very surprising.
- Extended PDI may thus be a reasonable measure of progress.

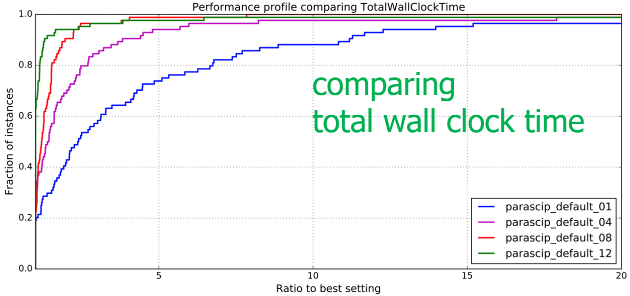


(a) ParaSCIP (b) SYMPHONY

Figure: The relationship between the wall clock time and the extended PDI.

Performance Profiles of Extended PDI and Wallclock



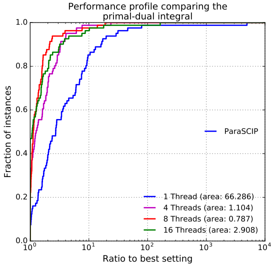


Outline

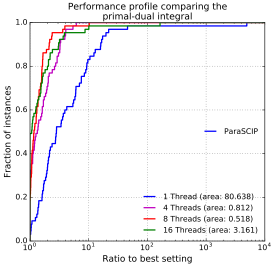
Sample Computational Results

- We have been experimenting with a number of ways of applying the ideas seen so far.
- In the following, we show results with the following solvers.
  - Gurobi
  - ParaSCIP [Shinano et al., 2013]
  - SYMPHONY [Ralphs and Güzelsoy, 2005]
  - ALPS [Xu et al., 2007]

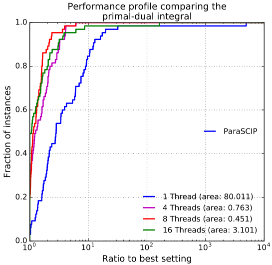
Performance Profile Using Extended PDI



(a) 18000 seconds limit



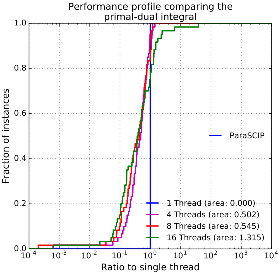
(b) 9000 seconds limit



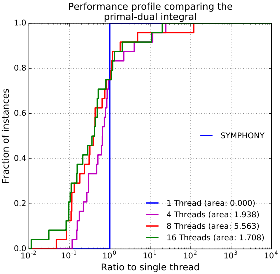
(c) 4500 seconds limit

Figure: Performance profile of PDI for ParaSCIP on MIPLIB2010.

Scalability Profile Using Extended PDI



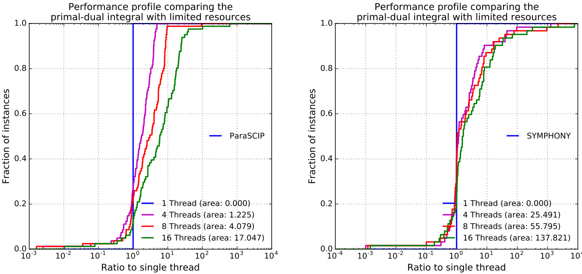
(a) ParaSCIP



(b) SYMPHONY

Figure: Scalability profile of the extended PDI

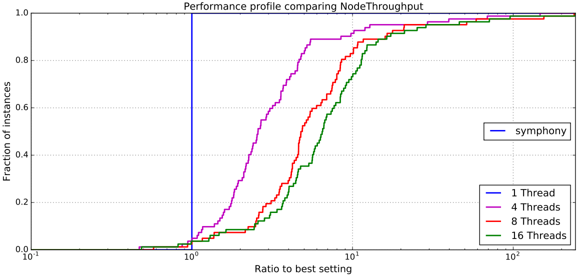
Scalability Profile with Fixed Compute Time

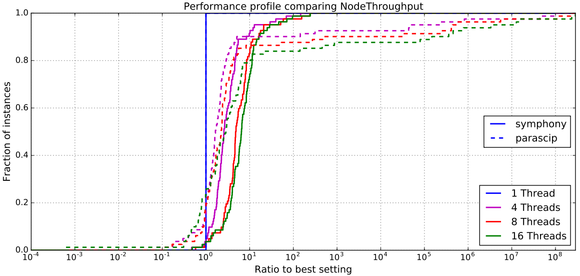


(a) ParaSCIP (b) SYMPHONY

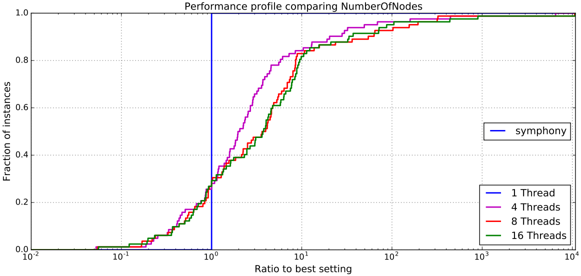
Figure: The scalability profile of PDI with fixed compute time.

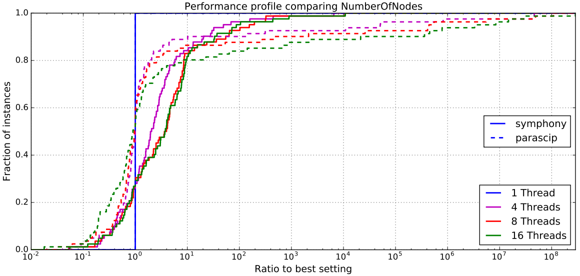
Node Throughput Scalability Profile



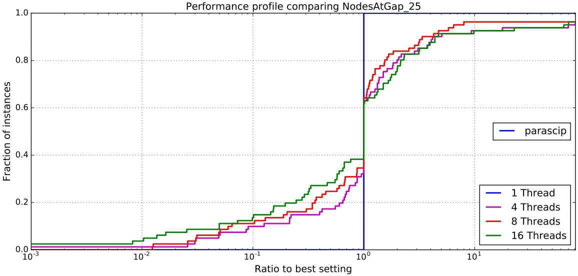


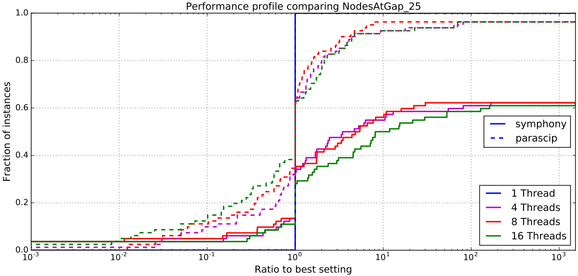
Number of Nodes Scalability Profile





Number of Nodes at Gap Scalability Profile





Outline

Conclusions

- What has been presented here is just a proposal meant to start a discussion within our community.
- These visualizations are not the end of the story, they may just indicate where to dig for more information.
- We are continuing with this long-term project to analyze the differences in the many existing approaches to parallel MIP.
- Feedback appreciated!
- For more details, see
  - Koch et al. [2012]
  - Ralphs et al. [2016]

References I

T. Berthold. Measuring the impact of primal heuristics. ZIB-Report 13-17, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, 2013.

T. Koch, T.K. Ralphs, and Y. Shinano. Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research*, 76:67–93, 2012. doi: 10.1007/s00186-012-0390-9. URL `http://coral.ie.lehigh.edu/~ted/files/papers/Million11.pdf`.

T.K. Ralphs and M. Güzelsoy. The symphony callable library for mixed-integer linear programming. In *Proceedings of the Ninth INFORMS Computing Society Conference*, pages 61–76, 2005. doi: 10.1007/0-387-23529-9_5. URL `http://coral.ie.lehigh.edu/~ted/files/papers/SYMPHONY04.pdf`.

References II

T.K. Ralphs, Y. Shinano, T. Berthold, and T. Koch. Parallel solvers for mixed integer linear programing. Technical report, COR@L Laboratory Report 16T-014-R3, Lehigh University, 2016. URL http://coral.ie.lehigh.edu/~ted/files/papers/ParallelMILPSurvey16.pdf.

Y. Shinano, S. Heinz, S. Vigerske, and M. Winkler. FiberSCIP – a shared memory parallelization of SCIP. ZIB-Report 13-55, Zuse Institute Berlin, 2013.

Y. Xu, T.K. Ralphs, L. Ladányi, and M.J. Saltzman. Computational experience with a framework for parallel integer programming. Technical report, COR@L Laboratory Report , Lehigh University, 2007. URL http://coral.ie.lehigh.edu/~ted/files/papers/CHiPPS.pdf.