コンピュータ将棋と並列化
～緻密ないい加減さ～

東京大学　横山大作

自己紹介

- 横山大作
- 東京大学生産技術研究所 助教
- 並列計算処理系、クラウド環境などの研究に従事
- 1999年、鶴岡らと「激指」開発スタート
  - トップレベルソフト
    - 世界コンピュータ将棋選手権4回優勝　など

探索問題

- 「砂漠からコインを探すような問題」に例えられる
  - 人間より機械の方が得意

将棋の場合…

- 「この一手だけが有効」という局面が比較的多い
  - 正解の一本道をたどらなければいけない
  - やみくもに探しても無駄

- とはいえ…

力こそ正義

http://commons.wikimedia.org/wiki/File:Bagger-garzweiler.jpg

- 計算機は12年で1000倍速くなる
  - 人間には直観しにくい性能向上

トレードオフの存在

- 「無駄のない読み」と「力任せの読み」

- コンピュータ将棋の歴史は、良いポイントを探す歴史
  - 手作業による絞り込み
    - コンピュータがあまりに非力だった。ぬかみそ的なコード。
  - 統計情報による絞り込み
    - 激指「実現確率打ち切り探索」
  - 機械学習によって評価関数を調整、なるべく絞らない
    - でも詰み探索だけは特別扱い

- ハードウェア・アルゴリズムの変化に伴いトレードオフ点が変化

力任せの読みのために: 並列計算

- ハードウェアの流れ: 並列性により高速化
  - CPU内部のコア数増大
  - クラスタ、クラウド環境

問題の並列化しやすさ

- Embarrassingly Parallel
  - 単純に問題を分割すればよいもの
  - SETI@home, Folding@home…
  - スケールアウト性が課題
- 計算に依存関係があるもの
  - ほとんどの問題
  - 行列計算、グラフ問題、…
  - アルゴリズムの工夫、通信の工夫などが必要
- 単純に並列に実行すると無駄が生じるもの
  - 将棋

将棋における並列化

- 「無駄のない読み」と「力任せの読み」のバランスが難しい問題
  - 問題の性質上、「一切無駄が生じない」並列化は難しい
  - 探索順序が効率に影響を与える
- 探索問題一般でも同じ課題が存在する（はず）

- 将棋プレイヤの並列化・分散化における手法を紹介

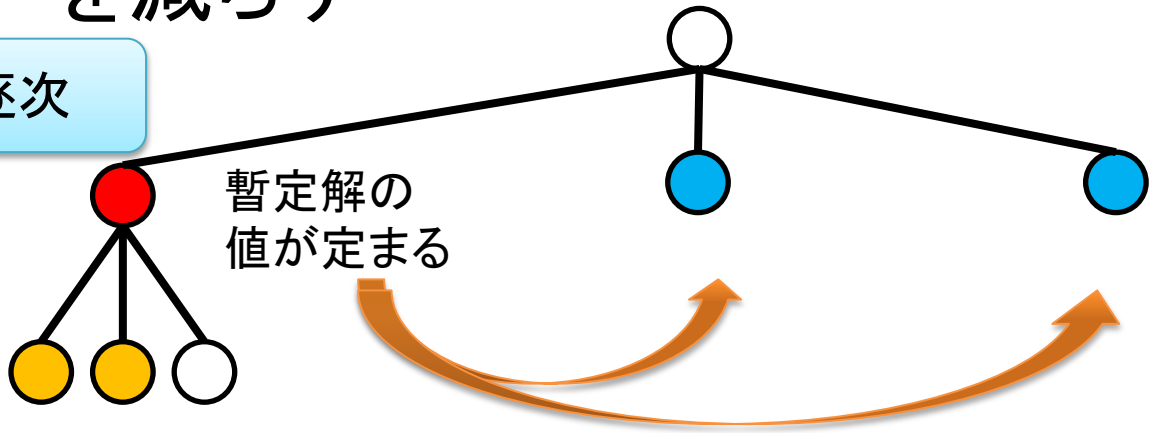
並列化で無駄を生み出す要因: 何が難しいか？

- 探索順序の違いから発生する無駄
  - アルファベータ探索による枝刈り
  - Transposition Table経由の枝刈り
- 粒度の違いによる扱いにくさ
  - スケジューリング実装の難しさ
- 並列実行できそうな部分がもともとは無駄な計算
  - 探索して初めてわかる「必須」
  - ゲーム特有のヒューリスティックが優秀

探索順序により発生する無駄:
アルファベータ探索起因

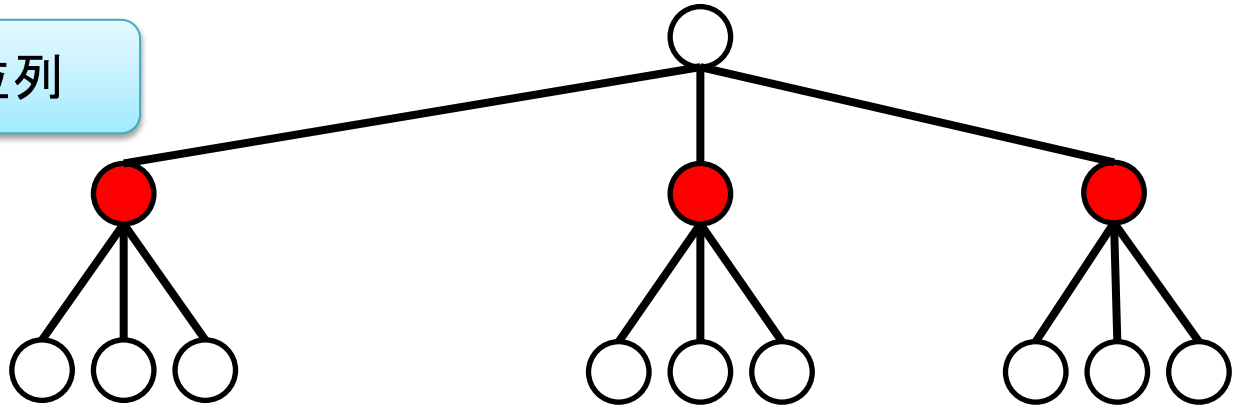
- 暫定解の情報を用いて枝刈りを行い、探索量を減らす

逐次



暫定解の値より良くなるなら相手方はそれを選ばないはず、という理由から枝刈りができる

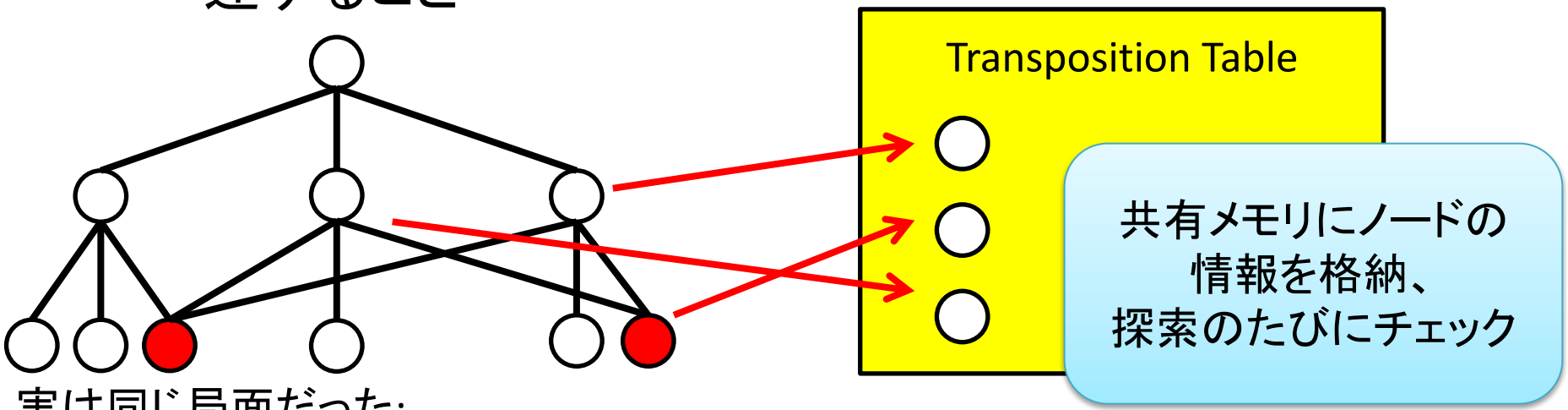
並列



情報不足で
枝刈りできない

探索順序により発生する無駄: Transposition Table起因

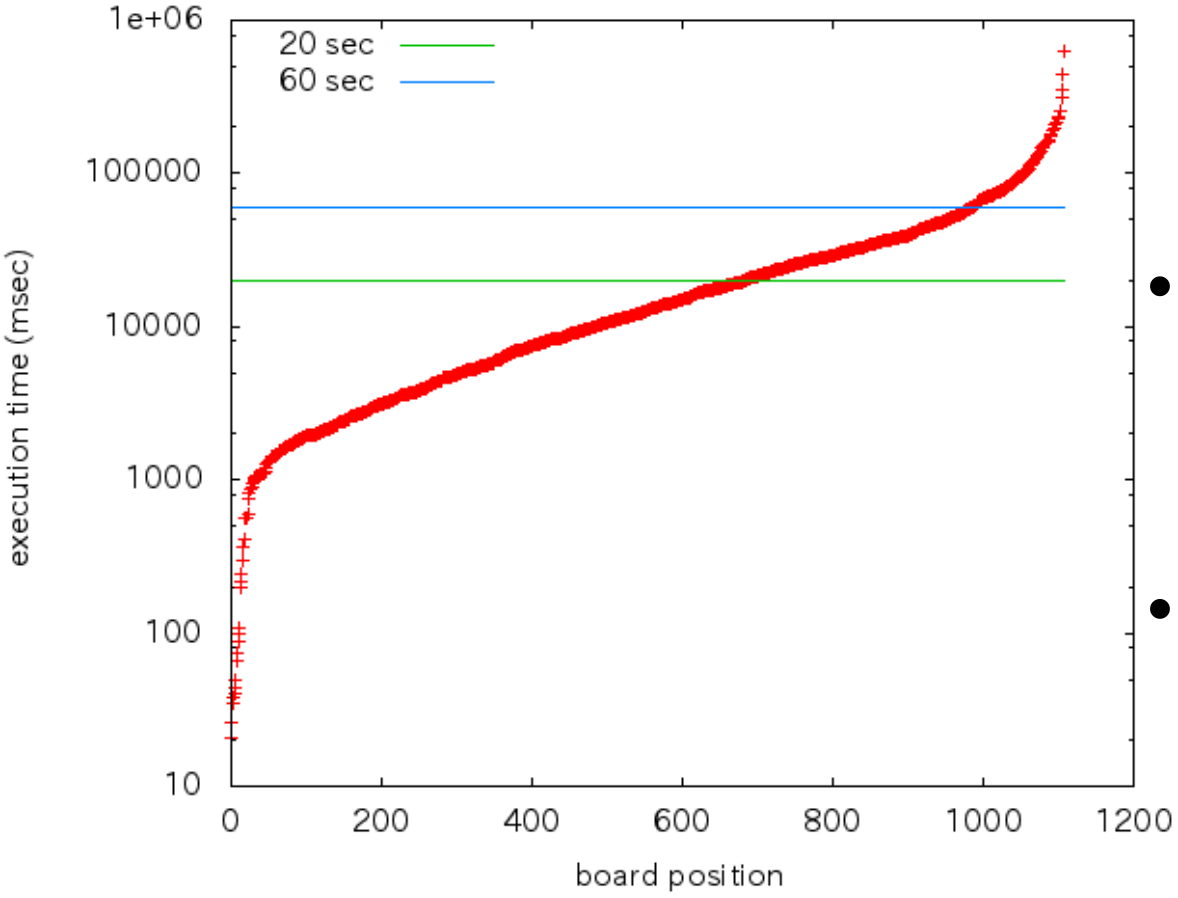
- 部分的な木の探索結果を保存する領域
  - Transposition: チェス用語。異なる手順で同じ局面に達すること



実は同じ局面だった:
別々に値を求めるのは無駄

Tableに書かれる前に
同じノードの探索を始めてしまうと無駄

粒度の違い

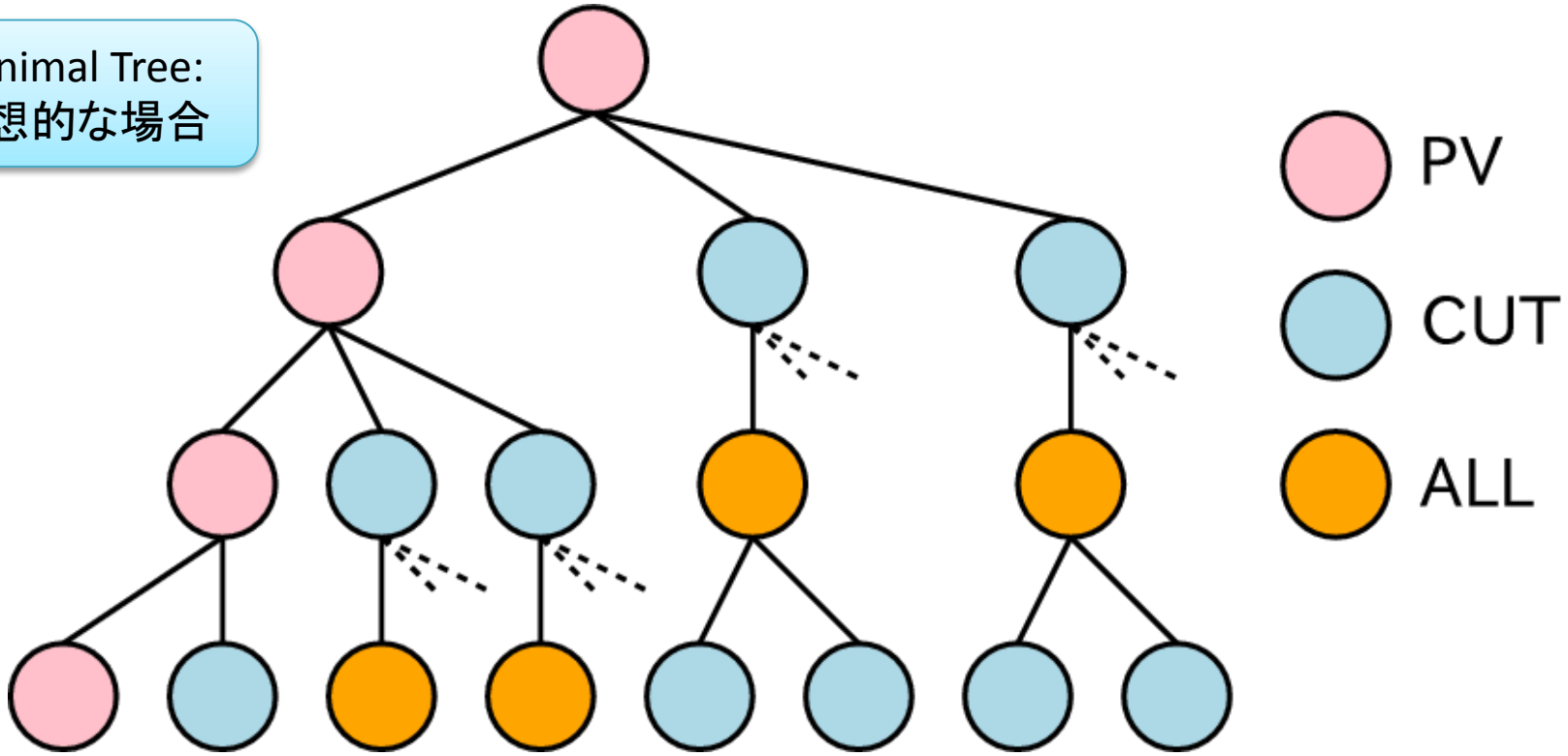


ランダムな実戦局面における所要計算時間

- 木探索の部分ごとに計算に必要な時間が大きく異なる
  - どこに合わせて実装したらよいか
- あらかじめ予測できない
  - 探索が終わって初めてわかる
- 静的なスケジューリングでは対応が難しい
  - 「自分の仕事が終わったら助けに行く」ことが必要

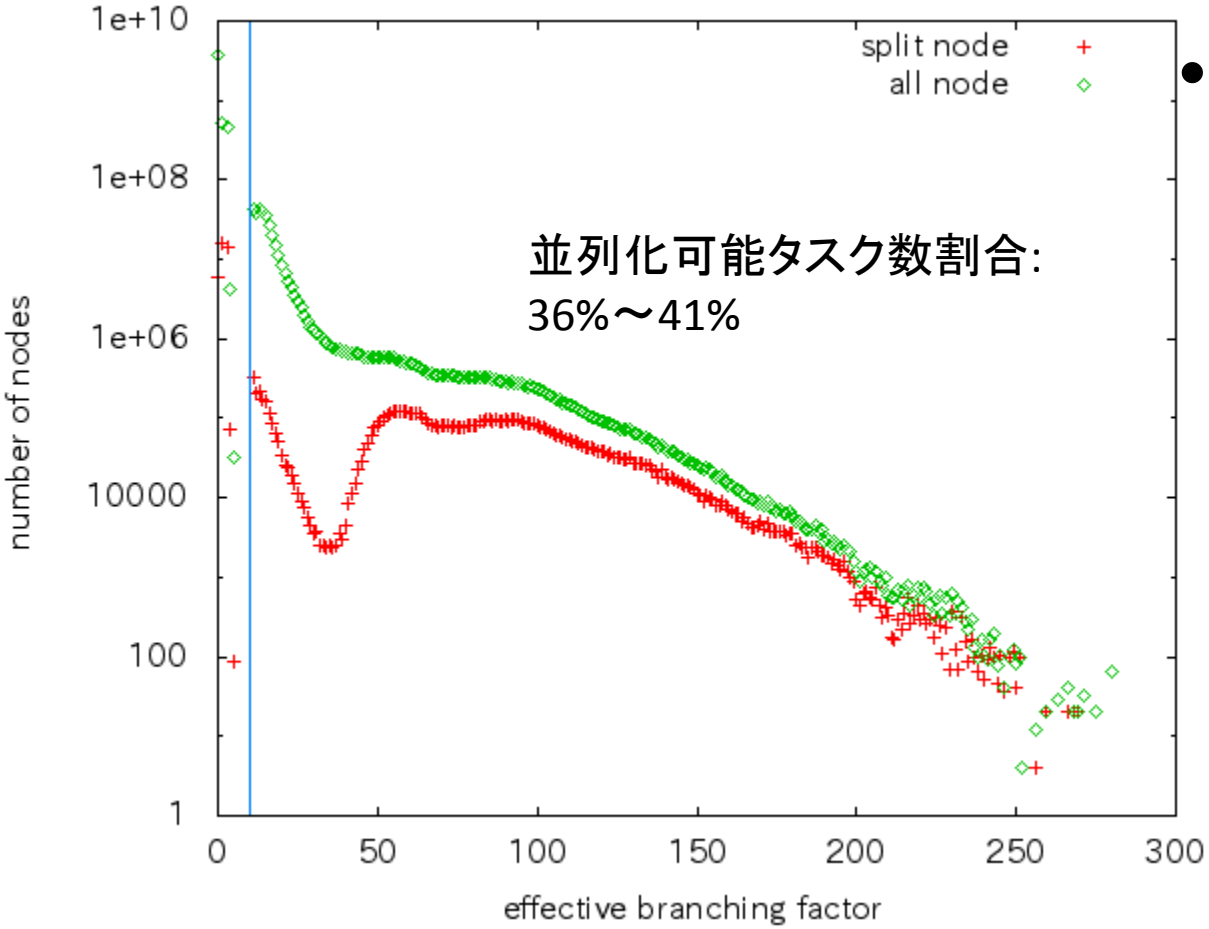
何が「必須」な計算か:ノードのタイプ分け

Minimal Tree:
理想的な場合



- PVは探索して初めてわかる
- 様々なテクニックにより、Minimal Treeに近い木が得られているのは救い

ゲーム特有のヒューリスティックが優秀

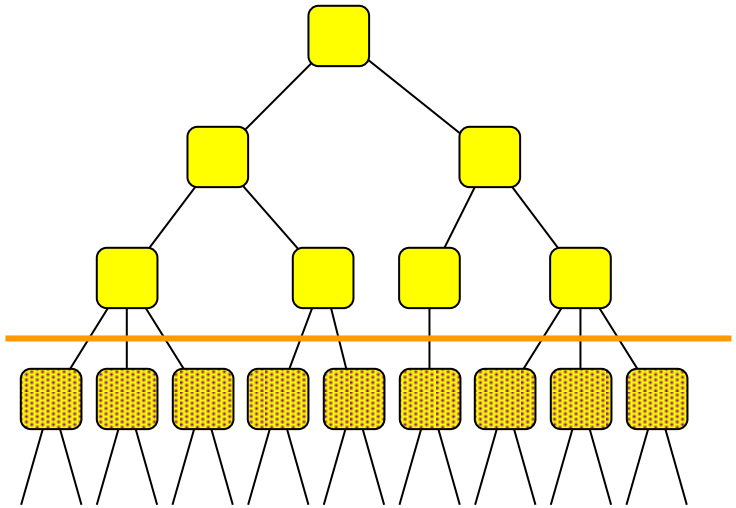


激指における実効分枝数(子供の数)の分布

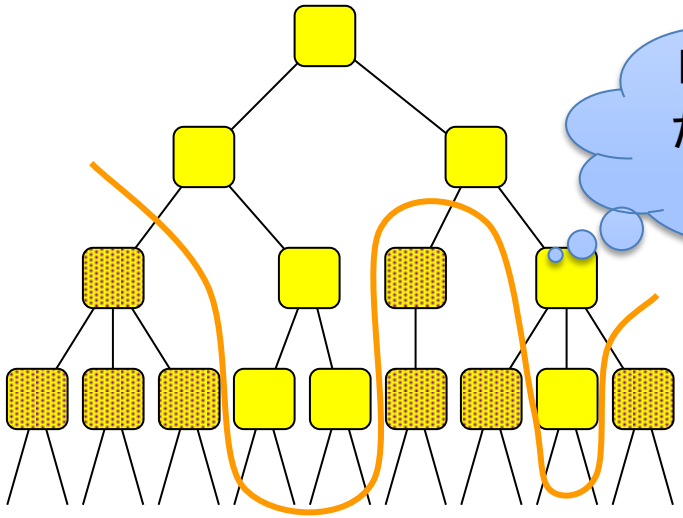
- 多くのノードが最初の数個の子供を調べるだけで終わる
  - 並列化できるときには相当数の子供を生成

実用的な並列化アルゴリズムの設計

- ある程度逐次探索の効率を残す
  - 探索順序をある程度逐次に近づける
- アプリケーションごとに性質が異なる
  - 激指: 選択的な読み(実現確率探索)。細く深く読むので粒度の差が大きい
  - Bonanza: 全幅探索が基本(枝刈りテクニックは多数使用)。より並列動作を追求した方が効果が出る可能性あり



全幅深さ打ち切り

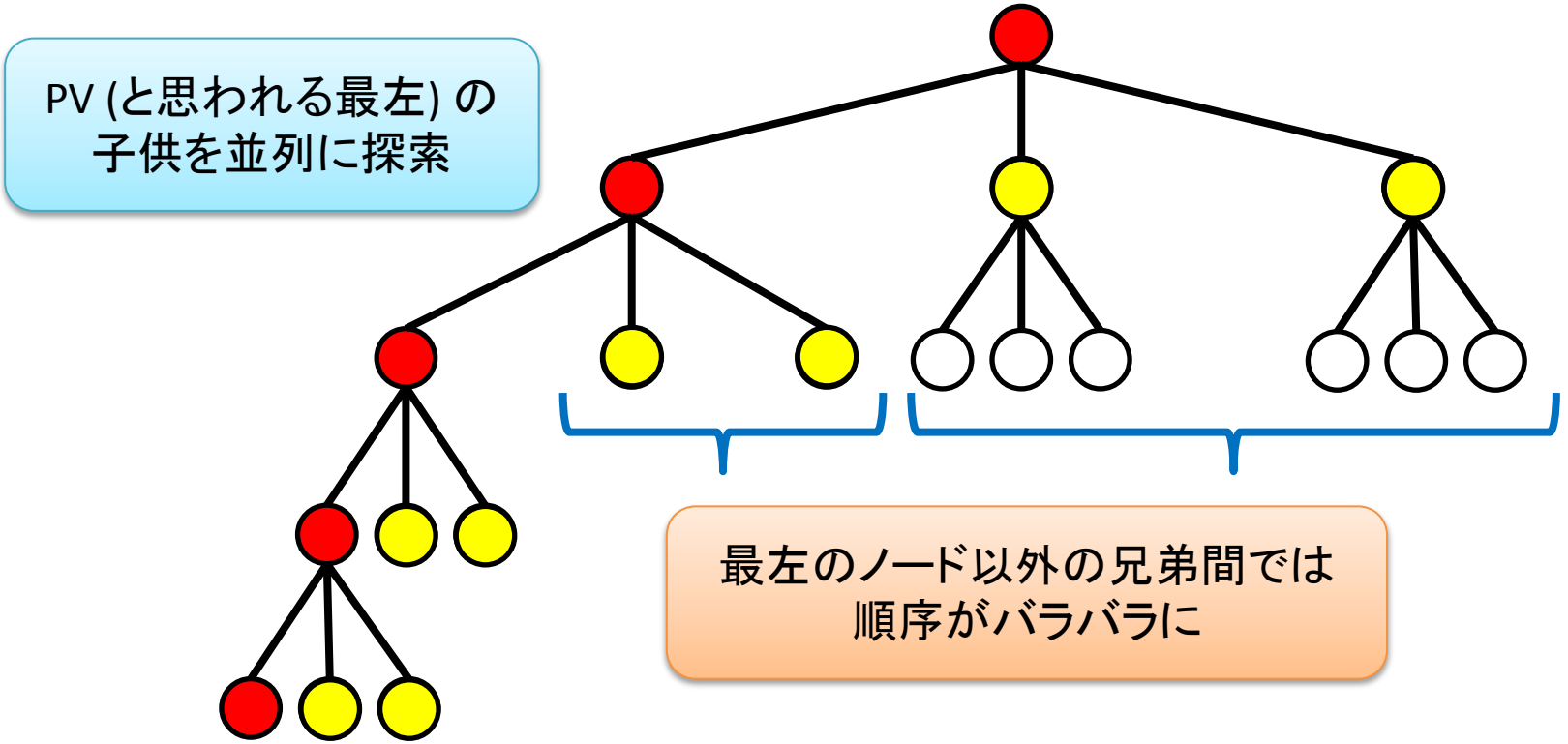


実現確率探索

並列スケジューリング

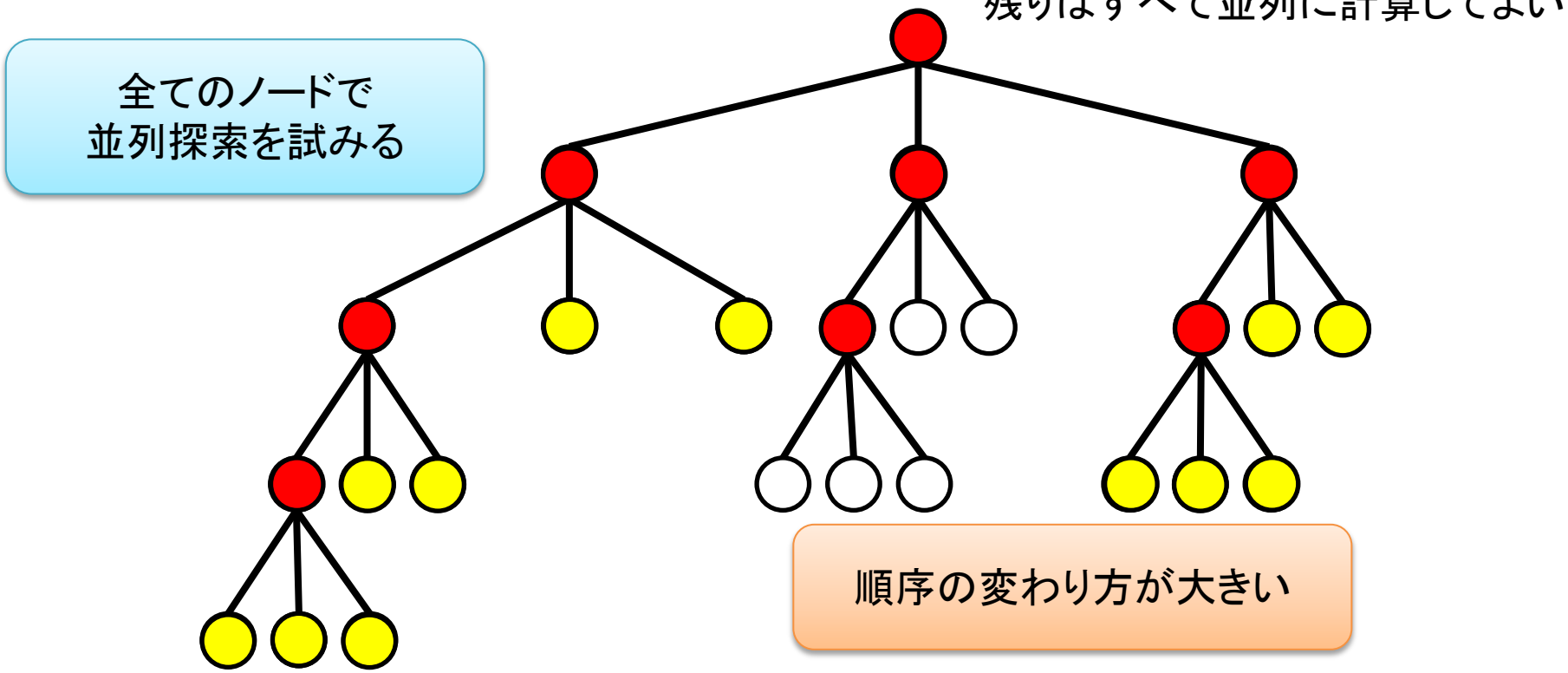
- どの部分を実際に並列に計算するのかを制御するアルゴリズム
  - 古くから多数研究あり
- 単に「CPUを暇にしない」ことを目指すだけならまだ簡単
  - ランダムワークスティーリング
- 都合よく順序を制御しようとすると問題が難しくなる
  - 実装面課題: 例) グローバルな通信が必要にならないか？
  - アルゴリズム面課題: 例) そもそも適切な順序があるのか？

PV-split



- 逐次の探索順序を比較的保存している
- 並列に実行できる部分が少ない

YBWC (Young Brother Wait Concept)

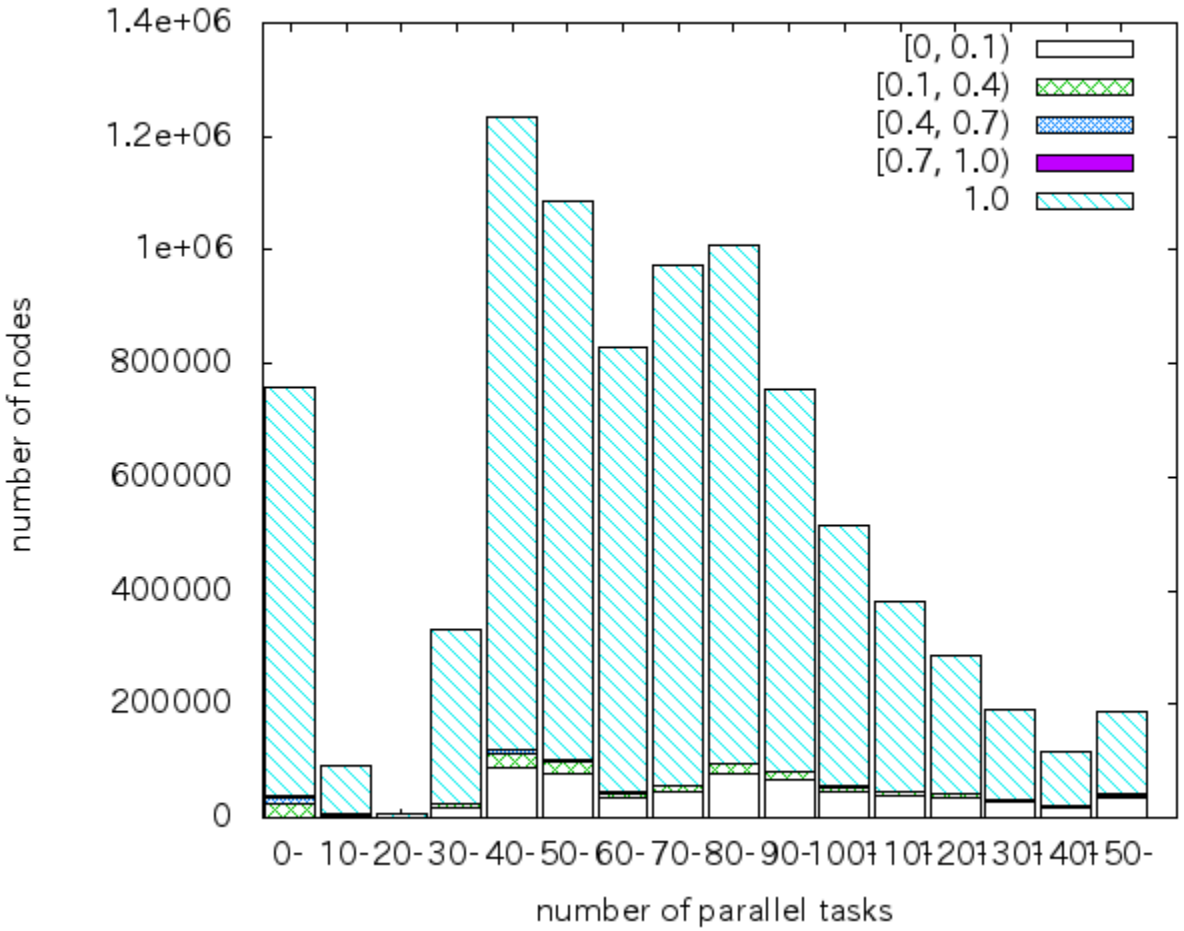


- 並列に実行可能な部分は増加
  - 見切り発車で探索を始める部分が多くなる
- 単純な制御 (必要な情報が少ない) で効果が出る

激指の並列化

- YBWCに近い動作
  - ヒューリスティックに基づいた数個の子供を逐次的に探索、カットできなければその後を並列探索
- 動的スケジューリング
  - 暇になったCPUが他のノードを助けに行く
  - 誰を優先的に助けるかは現状ランダム
    - 制約を加えても効果を発揮しない場合がある
    - 制御法は今後検討の余地あり

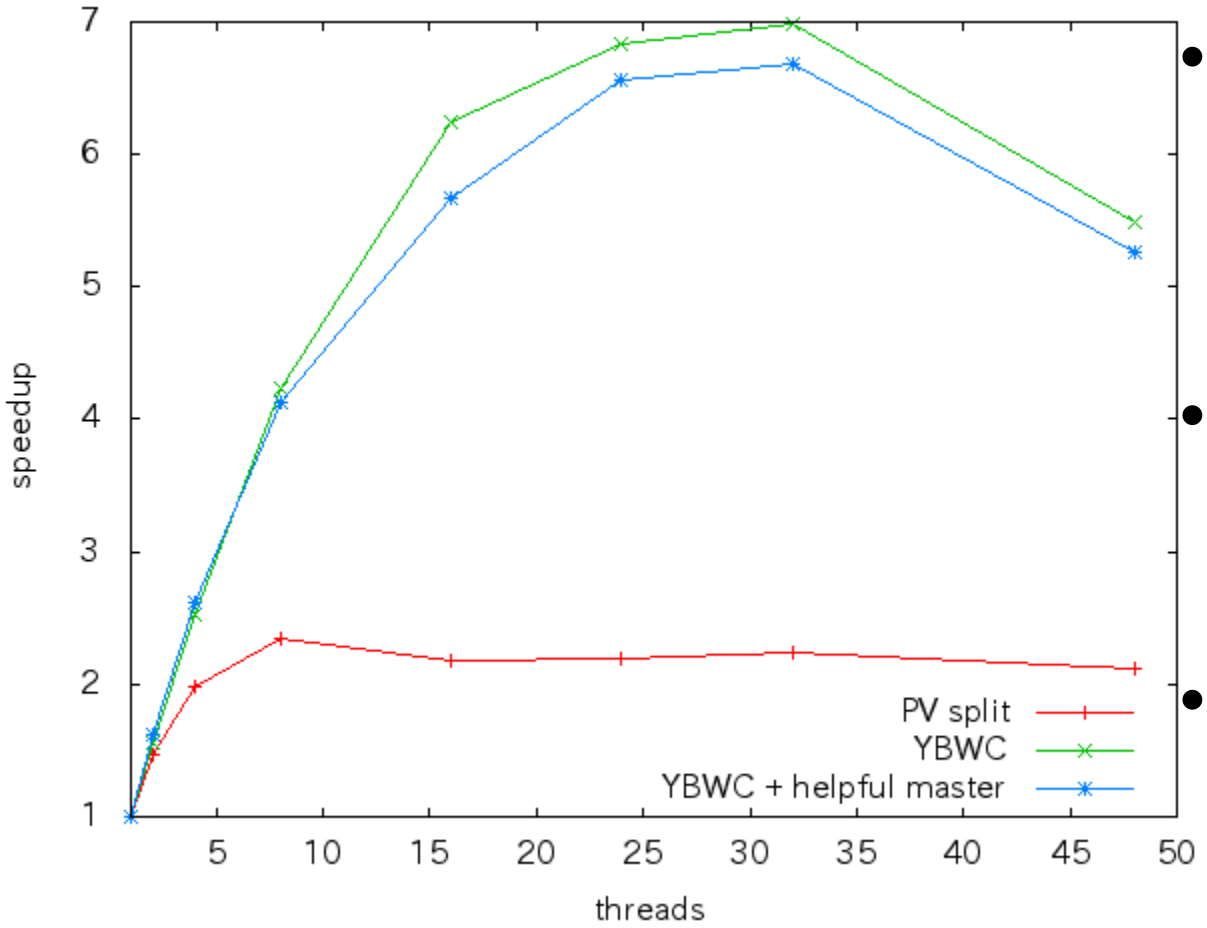
並列化対象ノードの性質



並列対象ノードの並列度分布と、
最終的に計算に必要だった割合

- 並列化することにしたノードは十分な並列度を持つ
- タスクの多くの割合が「必要」だったもの

並列化性能



共有メモリ上での並列化による実質速度向上率
（ある局面からの探索が終了するまでの時間の短縮率。
5回平均速度向上率、50問幾何平均）
Opteron 2.2GHz, 12 core × 4 sockets

- PV splitでは並列度が足りない
  - 8並列程度で頭打ち
- YBWCによって性能向上するが32並列程度で頭打ち
- スケジューリング制約を加えると(helpful master)、無駄な探索は減るが速度はやや落ちる

並列計算・分散計算

- 並列計算: 同一メモリ空間内での並列実行
  - (ハードウェア)共有メモリを利用して情報交換
- 分散計算: 分割されたメモリ空間の間での並列実行
  - いわゆる「通信」による情報交換
  - 大規模な計算は分散計算でしか実現できない

- 計算と通信の速度比が異なるだけ、本質的な違いはない
  - 実用的には大きな違いに見えている
  - プログラミングに必要なツールは異なる

大規模分散計算の難しさ

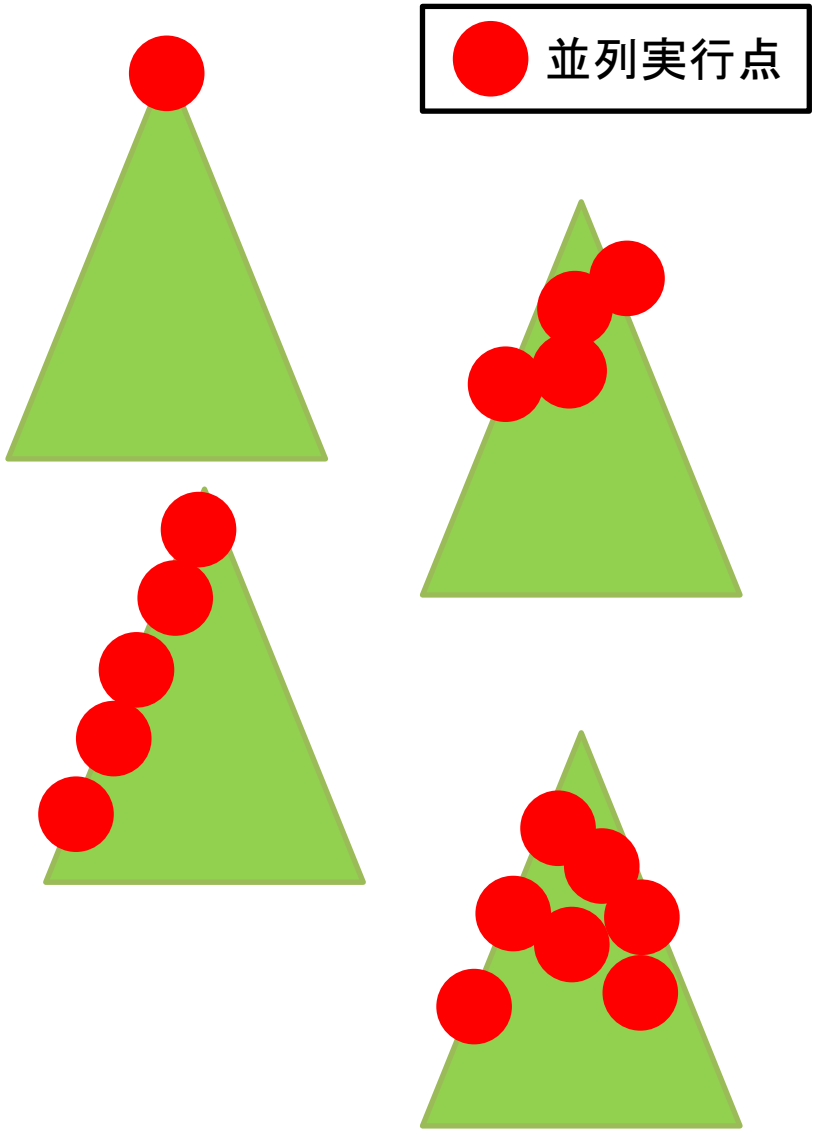
- 通信コストが大きい
  - 探索では、レイテンシの大きさが特に痛い
- 通信して分割する価値のあるタスクのサイズが大きくなる
  - 短い持ち時間の勝負では難しい
- Transposition Tableはほぼ共有不可能
  - 無駄な探索が生じる
- 大規模計算では耐故障性が必要

分散計算でもトレードオフ

- 「昔は良かった…」
  - CPUが非力で、相対的に分散計算が性能を出しやすかった
- CPU速度向上、通信コストが相対的に大に
- CPU台数の増加
- ノード内並列とノード間並列
  - 階層的な構成や、近い⇔遠いの違いなどが発生
- どのような設計が適切か、簡単には言えない

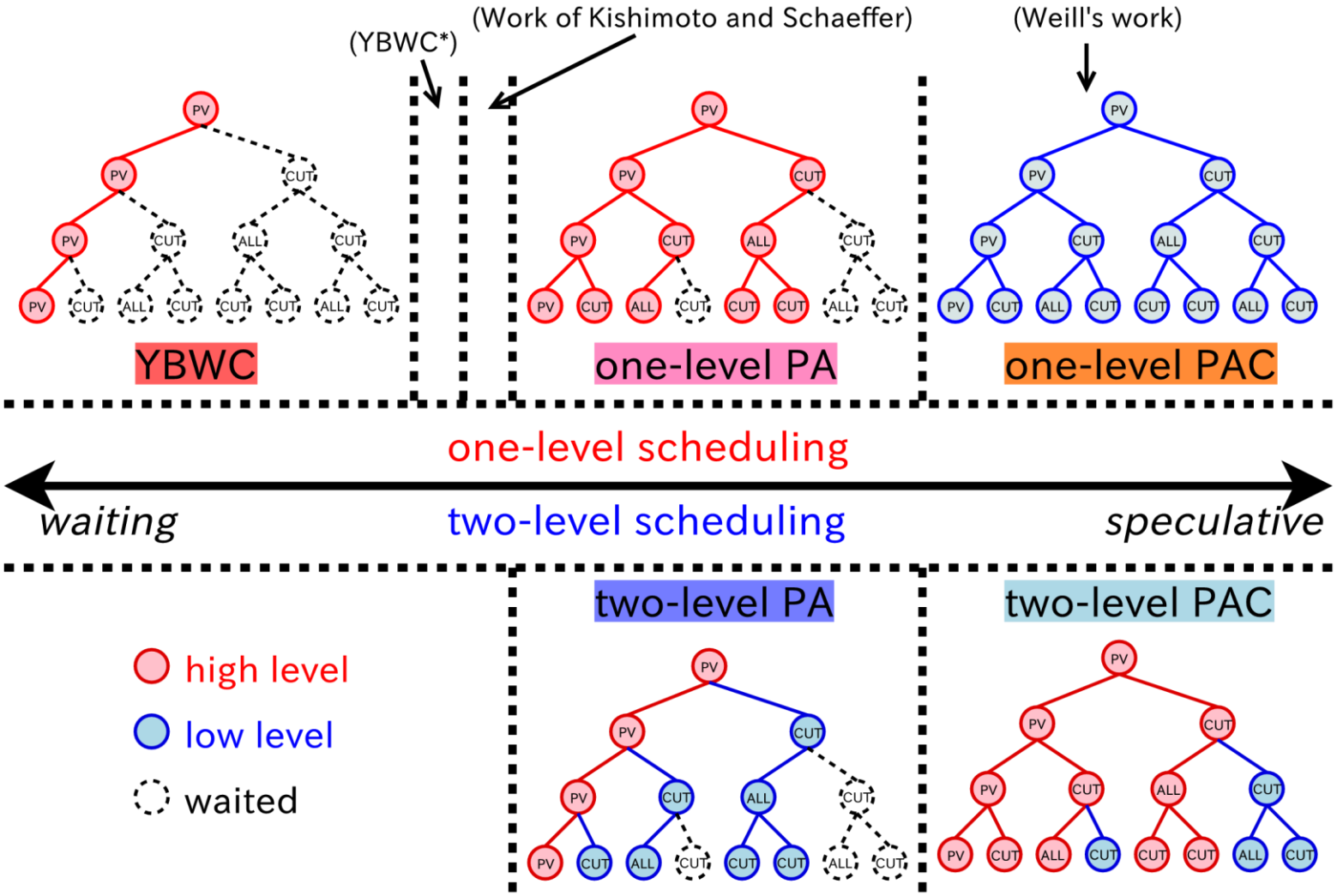
将棋における分散計算

- 合議 (Bonanza)
  - 少し乱数を加えた探索の多数決
- 静的な木 (GPS)
  - 開始時に少し時間をかけて木の分割を決定、以降は変化させない
- PV split (Puella α)
  - 親のノードの探索を投機的に始める工夫あり
- Two-level (浦ら)
  - YBWCによる探索
  - 探索待ちノードに優先度を付けることで性能向上

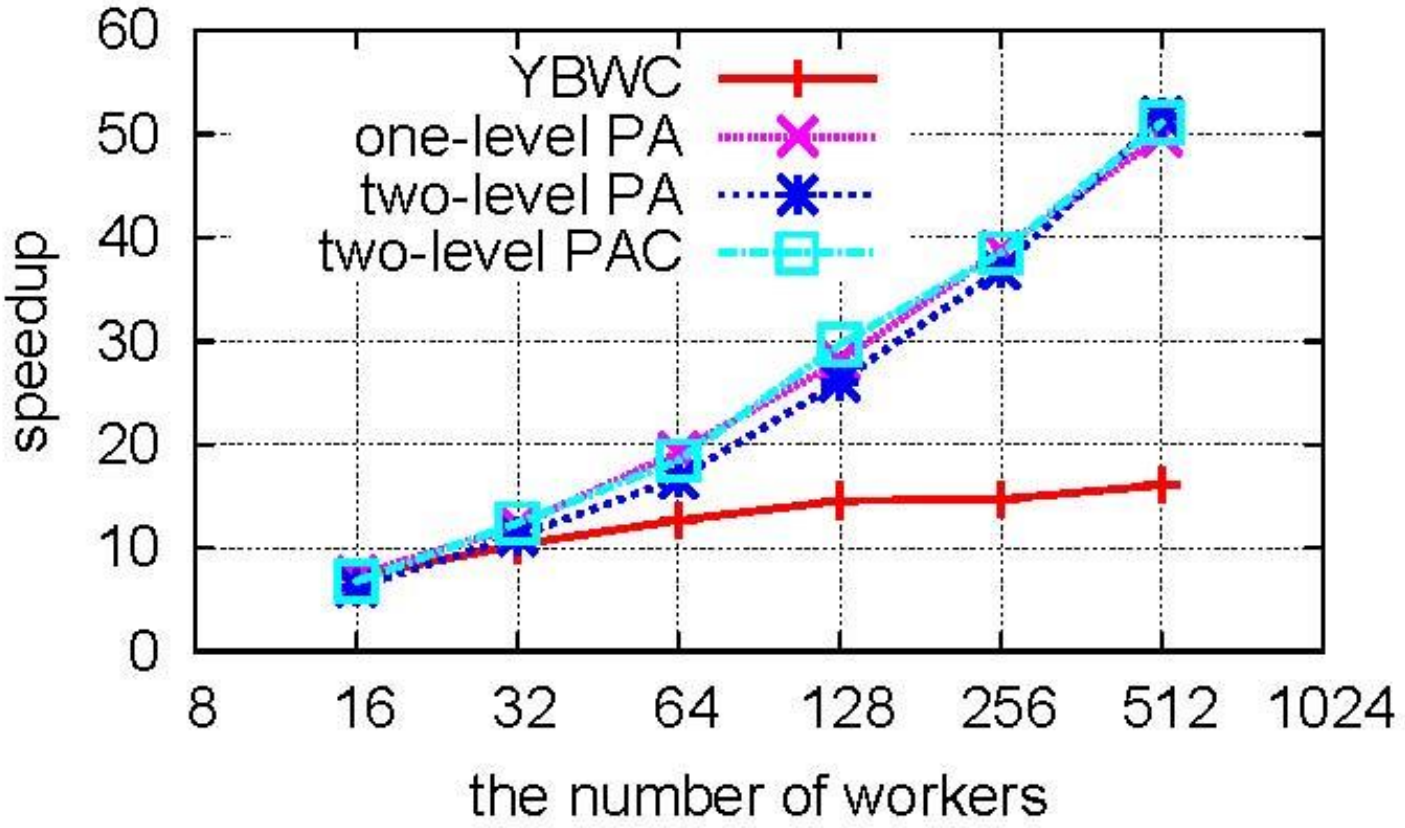


Two-level task scheduling [Ura et al. 2013]

- 大規模化に向けて並列実行できる部分を増やす



激指を用いた大規模分散探索



- スケジューリング改善でYBWCより良好な速度向上

耐故障性の追求

- 分散計算の大規模化により故障発生は不可避に
- 様々なレベルで対応の研究
  - 分散スナップショット
  - 耐故障を考慮したアルゴリズム
    - 多数決
    - TDSを用いた耐故障分散計算フレームワーク (野澤ら)

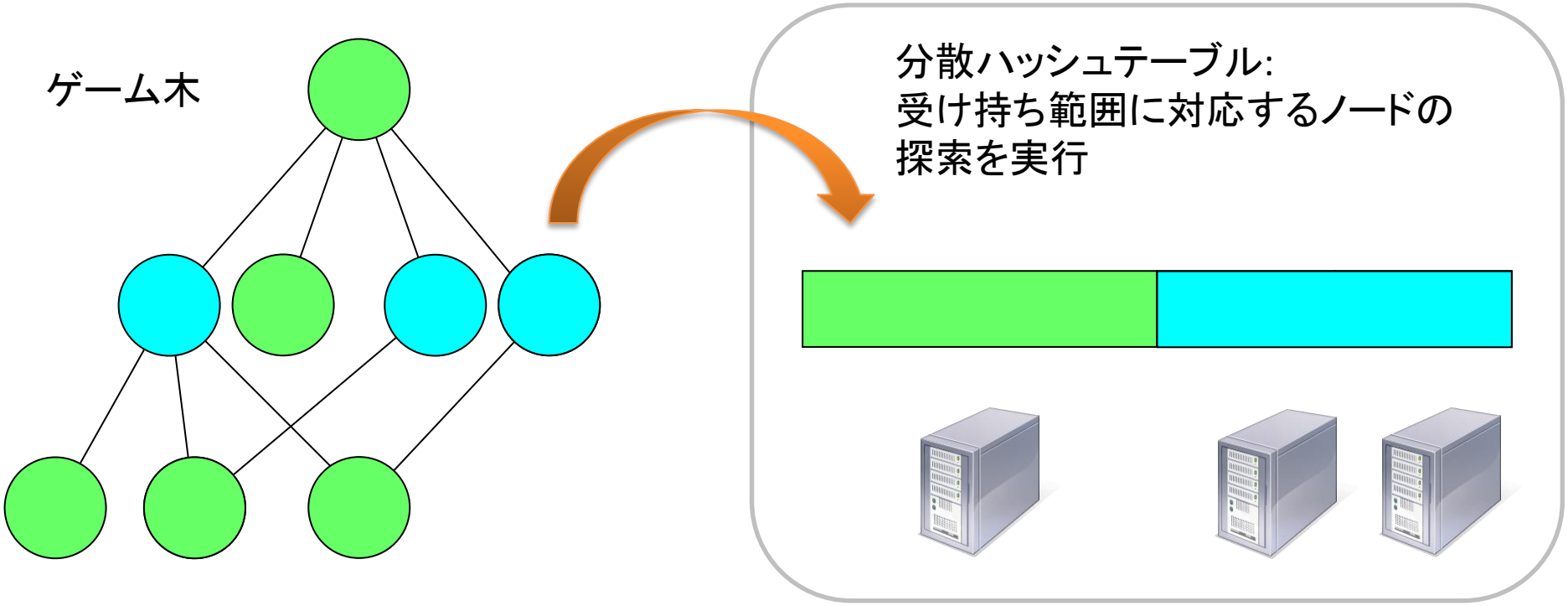
Transposition table Driven Scheduling（TDS) [Romein et al. 1999]

- Transposition Table を分散配置
- 局面のハッシュ値を担当するノードが計算
- Distributed Hash Table (DHT)に計算を付けたようなもの

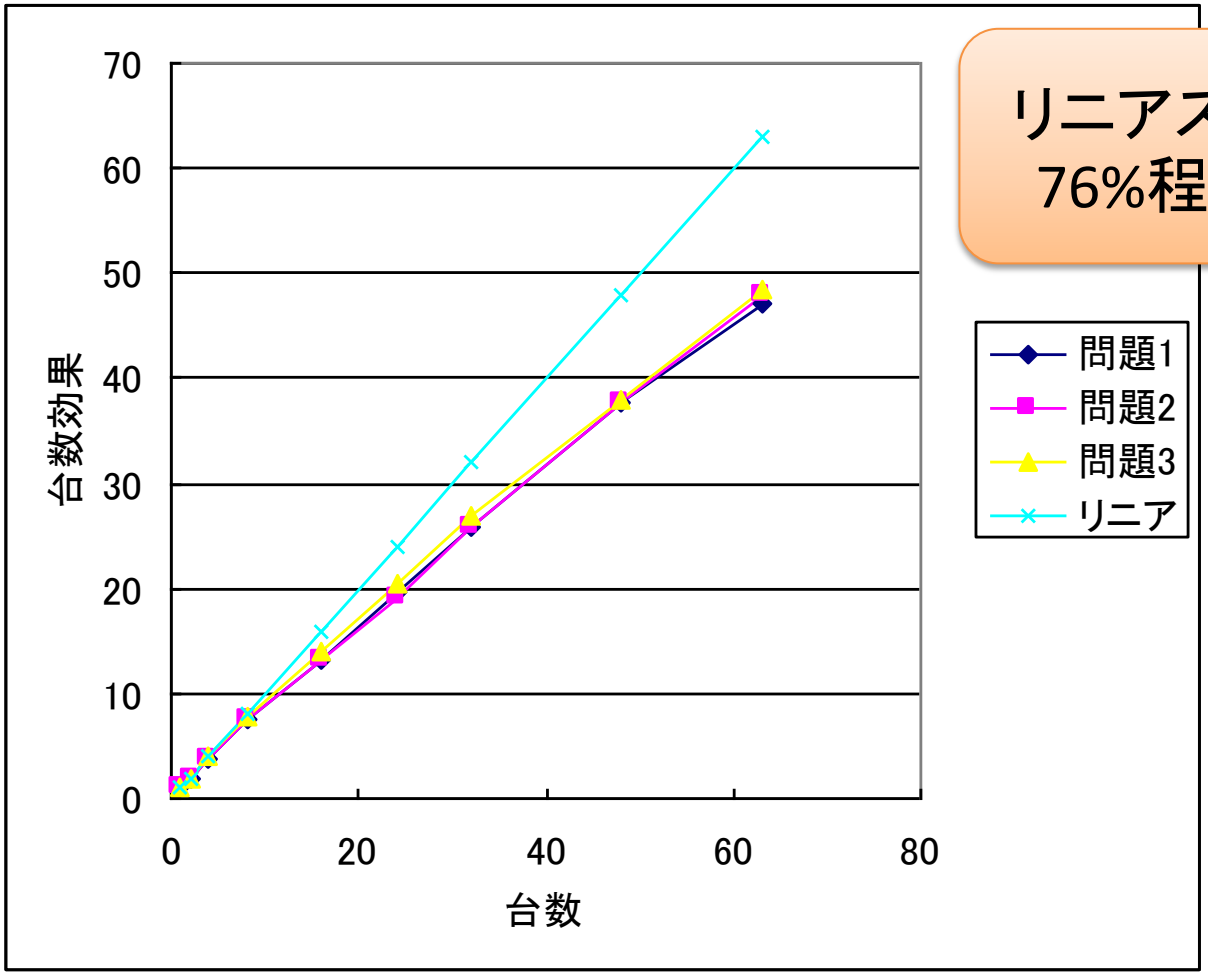
- ゲーム木への適用[Kishimoto et al. 2002]
- モンテカルロ木探索への適用[Yoshizoe et al. 2011]

TDSを用いた耐故障分散計算フレームワーク [野澤ら, 2006][横山ら, 2007]

- ルービックキューブの最短手順問題に適用
- 分枝限定法の反復深化探索

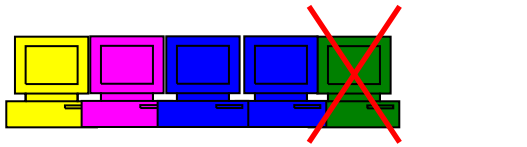


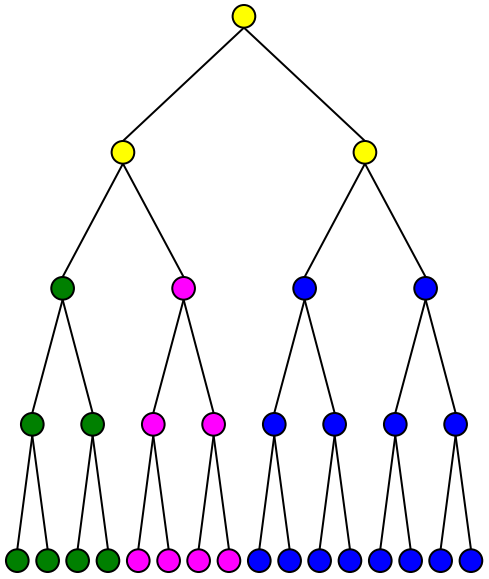
故障がないときの台数効果



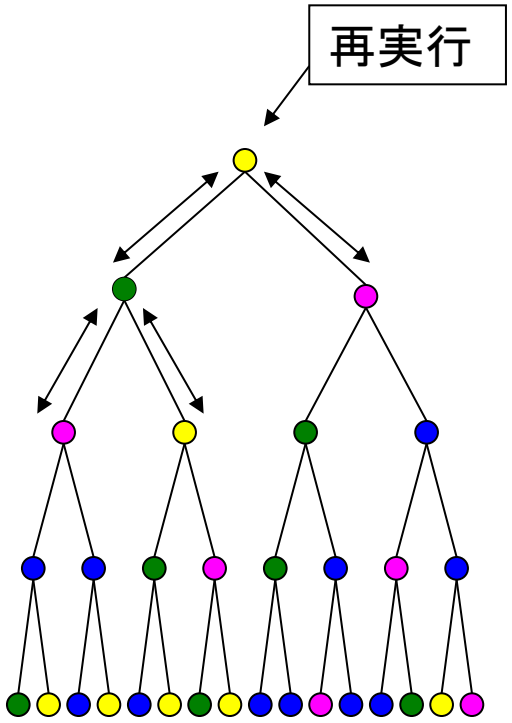
リニアスピードアップの
76%程度の速度向上

耐故障性の実現





探索部分木の分散

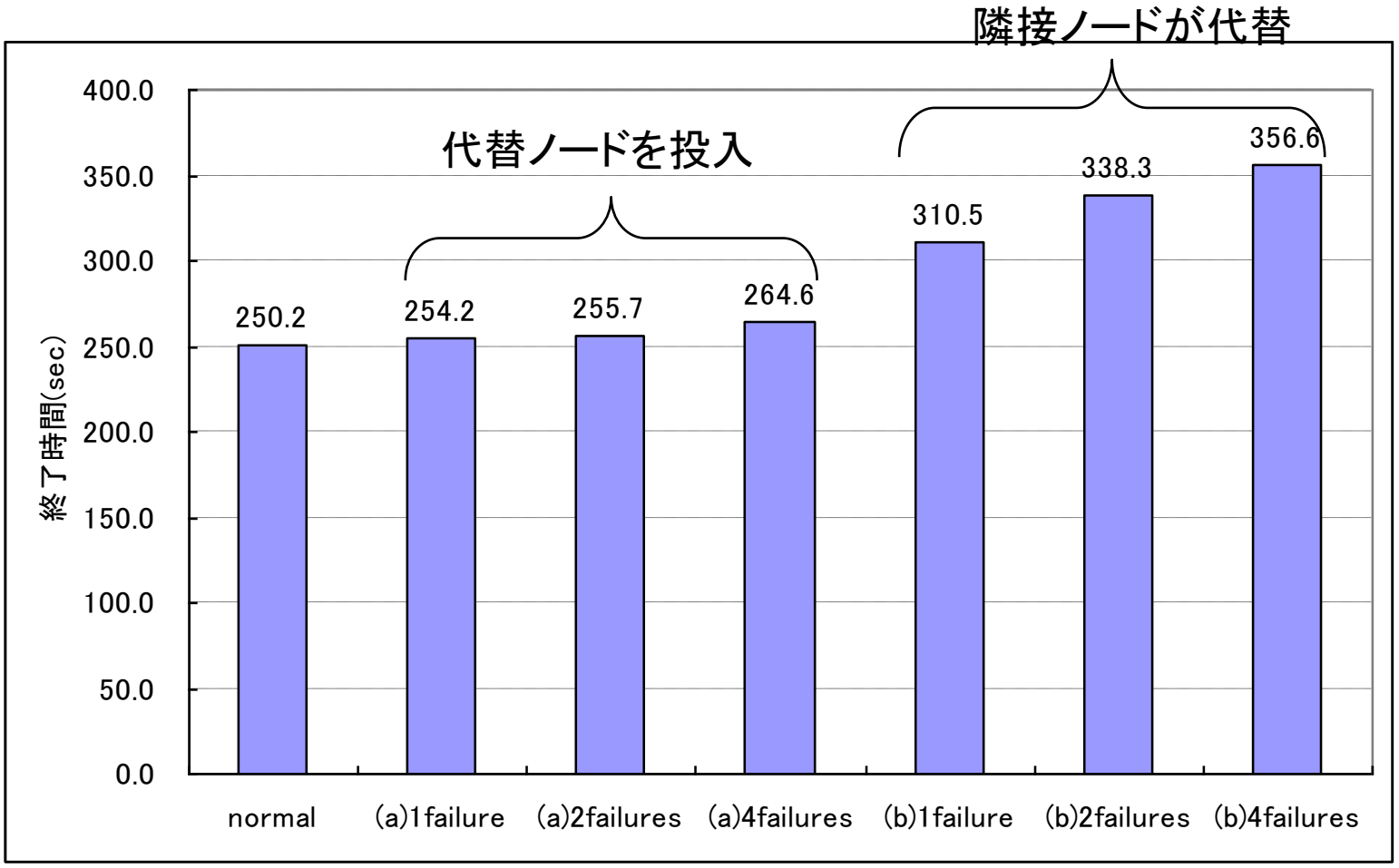


TDS

既知の部分解を
再利用することで
故障から迅速に復旧

細かくスナップショットを取ることに相当

故障挿入時の性能



故障発生時にも少ないオーバヘッドで探索続行可能

まとめ

- コンピュータ将棋における並列・分散計算適用への取り組みを紹介
- 「無駄のない読み」と「力任せの読み」のトレードオフが存在
  - 計算機環境、問題の性質、アプリケーションによって異なる最適点
- 多少無駄になってもよいと「いい加減」に構えている
- しかし緻密ないい加減さ
- 同様に難しい問題の参考になれば幸い

Questions?

参考文献

- 横山 大作.「激指」におけるゲーム木探索並列化手法, 人工知能学会誌 Vol.26, No.6, pp. 648--654, 2011.
- Akira Ura, Daisaku Yokoyama, Takashi Chikayama: Two-level Task Scheduling for Parallel Game Tree Search Based on Necessity, Journal of Information Processing, Vol.21, No.1, pp. 17-25, 2013.
- 横山 大作, 田浦 健次朗, 近山 隆. ハッシングに基づく大規模探索問題の耐故障分散処理手法, 情報処理学会論文誌: プログラミング, Vol. 48, No. SIG4 (PRO 32), pp. 1--13, 2007.
- 野澤康文, 横山大作, 近山 隆. 分散ハッシュ表に基づく大規模探索問題の耐故障並列化手法, 第58回 プログラミング研究発表会, 2006.

- 伊藤毅志, 小幡拓弥, 杉山卓弥, 保木邦仁. 将棋における合議アルゴリズム — 多数決による手の選択. IPSJ, Vol. 52, No. 11, pp. 3030–3037, 2011.
- 田中哲朗, 金子知適. コンピュータ将棋の不遜な挑戦 : 4. 大規模クラスタシステムでの実行 -GPS将棋の試み-. 情報処理, Vol. 51, No. 8, pp. 1008-1015, 2010.
- 伊藤英紀. コンピュータ将棋におけるクラスタ探索, 電子情報通信学会誌, Vol. 96, No.2, pp. 117-123, 2013.
- Akihiro Kishimoto and Jonathan Schaeffer: Transposition Table Driven Work Scheduling in Distributed Game-Tree Search, in Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence (AI ’02), pp. 56–68, 2002.
- Kazuki Yoshizoe, Akihiro Kishimoto, Tomoyuki Kaneko, Haruhiro Yoshimoto, Yutaka Ishikawa: Scalable Distributed Monte-Carlo Tree Search. SoCS 2011.