マルチスレッドで次を占うと

大津 金光@宇都宮大学

本日のお話

- マルチスレッド実行による高速化の観点から将来のプロセッサの姿を独断と偏見(と自分の願望も込めて)予想する

- 予想結果:
  - スレッドの投機実行支援機能の装備
  - プロセッサコアの実行状態監視機能
  - 自律的な実行時最適化を行うVMMが動作

ちょっと前まで

- シングルスレッド実行での高速化をひたすら追求
  - 動作クロック周波数の向上
    いかにハードウェア回路を速く動かせるか
  - 命令レベル並列性の活用
    いかに多くの命令を同時に実行できるか

消費電力の問題から動作クロック周波数が伸び悩み

命令レベル並列性の抽出に必要なコスト（ハードウェア資源とその消費電力）が急激に増大

最近では…

- 1チップ当たりの全体処理性能を重視
  - 1チップに複数のプロセッサコアを集積して性能向上
    マルチコアプロセッサの普及
- シングルスレッド性能は電力効率の追求へシフト
  - これまでと同等の性能をいかに電力を使わずに達成するか

マルチコアプロセッサ上でのマルチスレッド実行によりプログラムの高速化にシフト中

- スレッドレベル並列性の活用
  いかにチップ上のコアを働かせるか？

プロセッサコア数

- チップ上に集積されるコア数は増えていく
  - コア数の増加により増加した処理性能を支える足回り（チップ外とのデータ転送能力）の問題はあるが、ピーク性能を高めるために数は増える？

⇒増えたコアにどうやって有効な仕事をさせる？

- 複数のプログラムをそれぞれのコアに処理させる
- 一つの逐次プログラムを複数のコアで協調して処理
  必ずしも全てのコアを使い切る必要はない

シングルスレッド性能の向上

- 逐次プログラムの性能向上が決して不要になったわけではない
  - 依然として逐次プログラム性能の向上は必要
    Amdahl's Lawによりプログラムの性能が並列化困難な逐次処理部分により律速
- 増えたマルチコアを使って逐次プログラムを高速化する時代が来る！

マルチスレッド実行による高速化

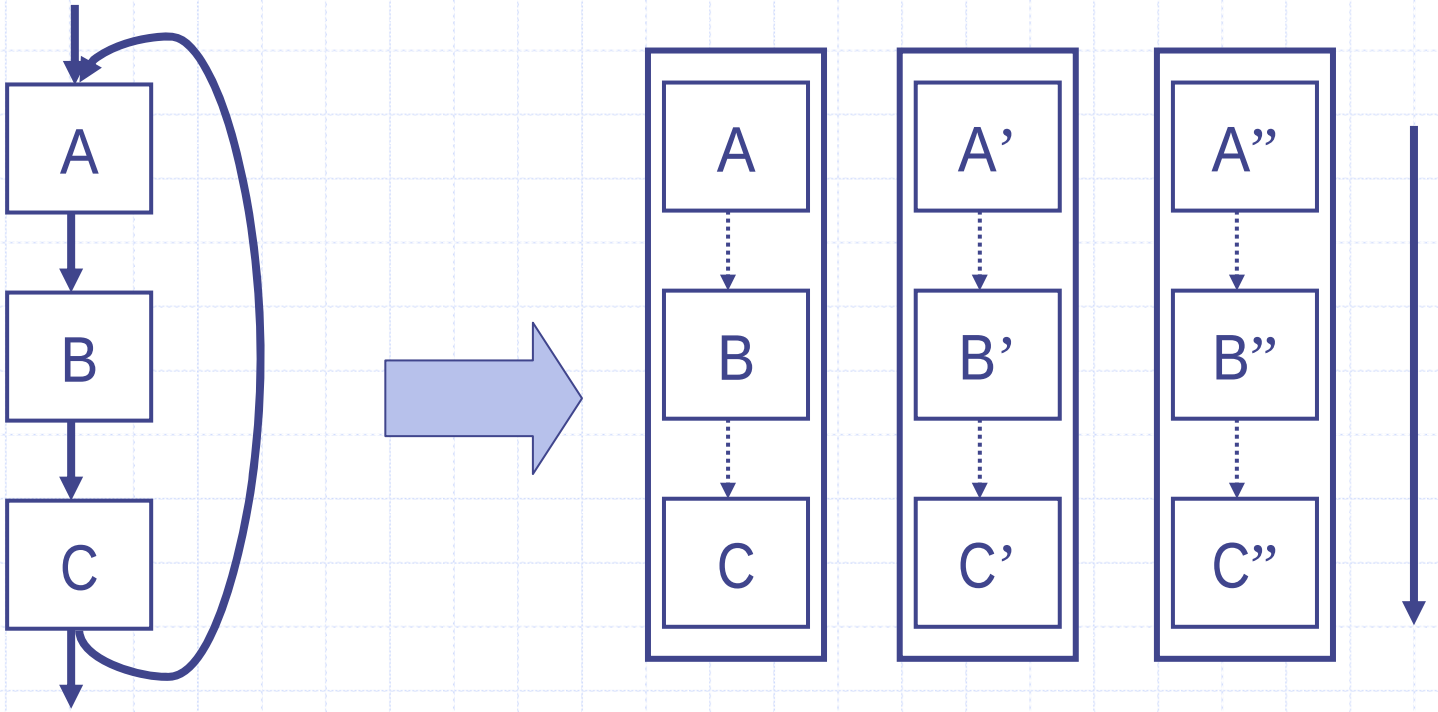
- 本来一つのスレッドで実行していたプログラムを複数のスレッドに分割して実行することで高速化
- いろいろな並列実行のやり方
  - ループレベル並列スレッド
  - 関数レベル並列スレッド
  - ヘルパースレッド

  など

ループレベル並列スレッド

- ループの各イテレーションを別個のスレッドとして並列実行
  - ループ終了するかどうかが直前のイテレーションの結果に依存する場合、先行的に次のイテレーションを実行しておいて後で判定
  - 直前のイテレーションの結果を予測して先に実行しておき、予測が正しかったかどうかを後で判定

ループレベル並列スレッド

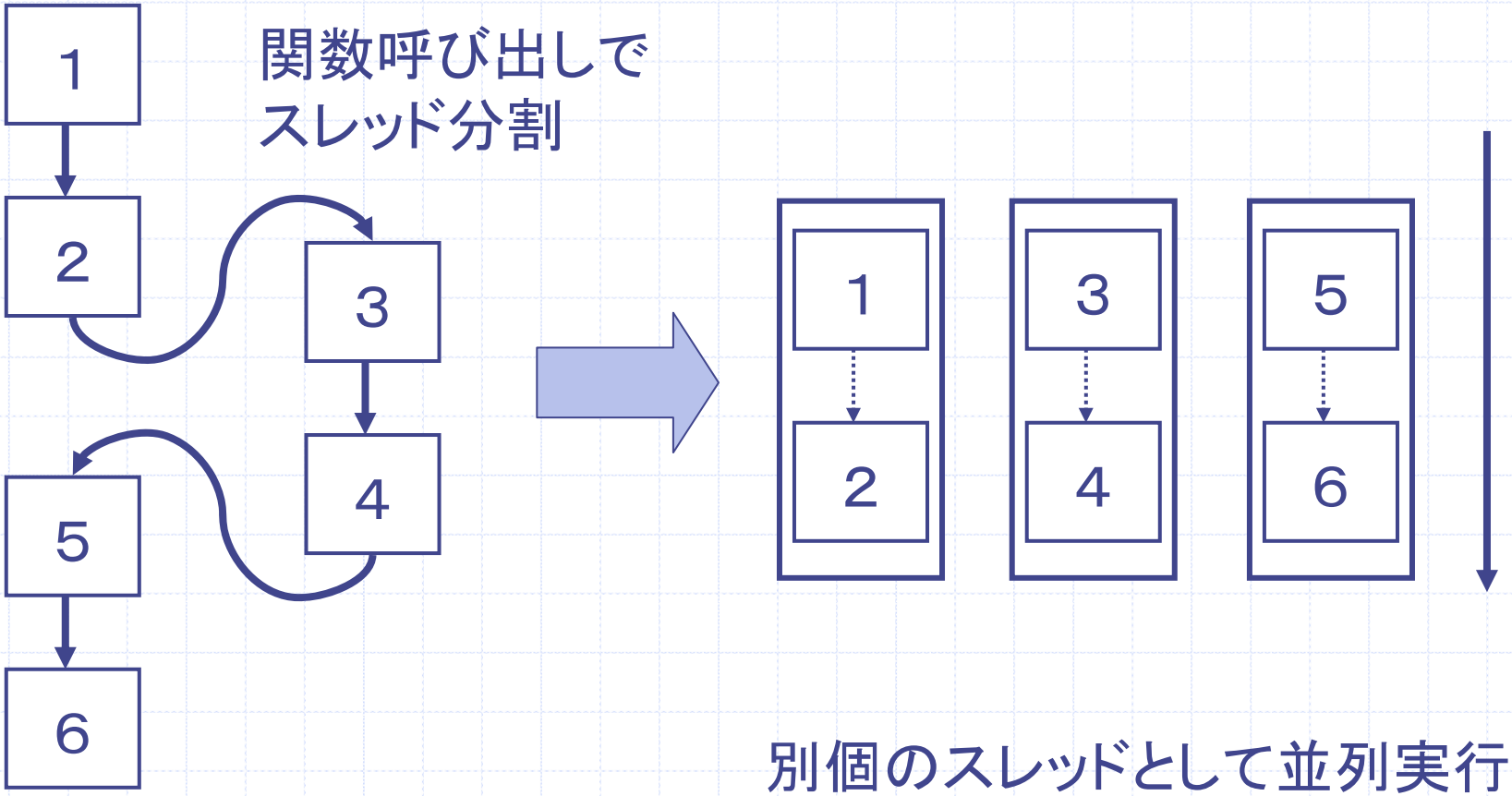


ループの各イテレーションを
マルチスレッドで並列実行

関数レベル並列スレッド

- 関数呼び出しを含むプログラムコードで呼び出し先の関数を別スレッドとして呼び出し元と並列に実行
  - 関数呼び出し時の引数や返り値を予測し先に実行しておき、予測が正しかったかどうかを後で確認

関数レベル並列スレッド



ヘルパースレッド

- 処理の本体を実行するスレッドの前に本体処理がスムーズに進行できるように手助け（ヘルプ）をするスレッドを別に走らせる
  - ヘルパースレッド側でプリフェッチすることでメインスレッドでのキャッシュミス低減
  - ヘルパースレッド側で本体計算の一部を先行実行しておくことでメインスレッドでの処理量の削減

シングルスレッド on マルチコア

- 通常の非投機的な並列実行では並列化が困難
  ⇒投機的な手法の導入
  　先にとりあえず実行しておき、後で必要のない実行の結果を破棄

- 実行結果の破棄に備えた投機的データの管理を効率的に支援する機構が装備される
  ⇒プロセッサコアの備えるロードストアバッファの拡張？

プロセッサコアの使い方

- 実行の確実性（投機成功率）が高い場合は高速化
  ⇒プロセッサコアを投機処理に活用

- 実行の確実性が低い場合は投機処理は無駄
  ⇒無駄な処理をしないことで計算資源浪費を回避
  （他の有効な処理にまわす or 停止して電力消費を抑制など）

プログラムに応じたスレッド実行

- プログラムによって適切なスレッド実行方式は異なり、適切な方式を選択する必要
  - 例えば、実行の確実性が高いのか低いのか

⇒実行時の挙動に冠する情報を基に判断
コア内部の実行時の挙動をそのコアの外側から監視できるような機構

実行時の状態監視

- 最近のプロセッサの持つPerformance Counterやデバッグ支援機能を発展させた実行時の状態監視機能

  ⇒性能に関わる各種イベント回数（投機成功回数やキャッシュヒット率など）をリアルタイムでカウント

  - あらかじめ定めた条件が成立したら割り込み等で通知（通知先は自分か他人か）

    投機失敗の機会が増えたら投機処理をしないように方針変更

実行時最適化

- 実行時の状態が把握可能になるとその情報を活用した実行時最適化が流行る
- コア数増加により増えた計算資源の一部を実行の最適化処理に割り当てることで大局的な処理の効率化
  - 例えば、シングルスレッドプログラムの高速化に割り当てて意味のあるコア数はプログラムによって異なる

    ⇒割り当てコア数の調整を自律的制御

VMMの発展

- 最近はVMM（Virtual Machine Monitor)が注目を浴びている
  - OSよりも下層で動作しプロセッサハードウェア資源を仮想化
  - 今後は実行時の最適化機能が取り込まれる
  - CMS (by Transmeta)のように実行時コード変換を行いながら上位で動作するソフトウェアコードをコア数に応じた最適化を行う可能性

まとめ

- コア数はピーク性能を上げるためにも増加
- シングルスレッドプログラムを複数のコアで高速化するためのスレッド投機を支援する機能
- コアの実行時の挙動を外側から観測するための機能
- 自律的な実行時最適化を行うVMMが動作

おしまい

ご静聴ありがとうございました