マルチプロセッサ

並列処理

並列処理

- 並列処理
  - 複数のCPUを用いて，
    高速に動作するコンピュータを実現するためのアプローチ

並列処理の効率

- 単一プロセッサによる実行時間 : T_1
- N個のプロセッサの並列処理による実行時間 : T_N
- 並列化できない処理の割合を α とした場合の T_N

$$T_N = \alpha T_1 + (1-\alpha)\frac{T_1}{N}$$

- 速度向上比

$$S_N = \frac{T_1}{T_N} = \frac{T_1}{\alpha T_1 + (1-\alpha)\frac{T_1}{N}} = \frac{N}{1+(N-1)\alpha}$$

アムダールの法則

- 速度向上の限界

$$\lim_{N\to\infty} S_N = \lim_{N\to\infty} \frac{T_1}{\alpha T_1 + (1-\alpha)\frac{T_1}{N}} = \frac{1}{\alpha}$$

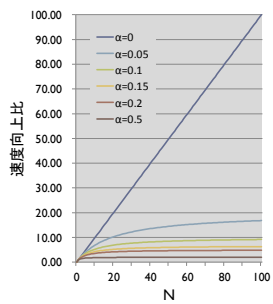
いかに並列度を上げても，性能は，
並列化できない部分の割合 α で
制限される．

速度向上比

- 速度向上比

$$S_N = \frac{T_1}{T_N} = \frac{T_1}{\alpha T_1 + (1-\alpha)\frac{T_1}{N}} = \frac{N}{1+(N-1)\alpha}$$

α は
並列化できない処理の割合



並列処理の分類

並列処理の分類

- 並列処理の分類
  - 命令流・データ流に基づく分類
  - メモリ共有方法に基づく分類
  - 結合の度合いに基づく分類
    （次ページ以降，各分類方法について説明）

命令流・データ流に基づく分類（1）

- 命令流・データ流に基づく分類（M.J.Flynn, 1966）
  - 命令流およびデータ流が，単一であるか複数であるかという点に着目した分類方法．
  - 以下の4種類に分類される．
    - SISD (Single Instruction stream, Single Data stream)
    - SIMD (Single Instruction stream, Multiple Data stream)
    - MISD (Multiple Instruction stream, Single Data stream)
    - MIMD (Multiple Instruction stream, Multiple Data stream)

命令流・データ流に基づく分類（2）

- SISD (Single Instruction stream, Single Data stream)
  - 1度に1つの命令を実行し，1つの命令で1つのデータを処理するような従来型のフォンノイマンアーキテクチャ．
  - スーパースカラやVLIWは，内部で命令が並列処理されているが，システム全体から見ると，命令の流れは単一であり，SISDに分類される．
- SIMD (Single Instruction stream, Multiple Data stream)
  - 各命令は単一の処理を指定するが，その命令が同時に多くのデータに対して適用されるようなアーキテクチャ．
  - 科学技術計算（行列計算）やグラフィックス（画像処理）の計算などにおいて，顕著な高速化が実現される．

命令流・データ流に基づく分類（3）

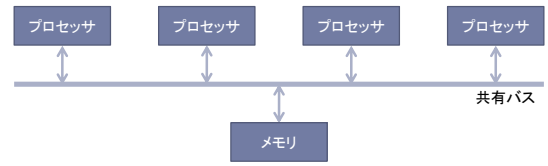
- MISD (Multiple Instruction stream, Single Data stream)
  - 一般に，これに該当するアーキテクチャは存在しない．
- MIMD (Multiple Instruction stream, Multiple Data stream)
  - 各プロセッサが同時に独立した処理を行うアーキテクチャ．
  - 複数の独立したプログラムを同時に実行することができる．
  - 現在の並列計算機は，ほぼすべてMIMDに属する．

メモリ共有方法に基づく分類（1）

- メモリ共有方法に基づく分類
  - メモリを共有するか否か，あるいはどのようにメモリを共有するかという点に着目した分類方法．
  - 以下の3種類に大別される．
    - UMA (Uniform Memory Access model)
    - NUMA (Non Uniform Memory Access model)
    - NORA (NO Remote memory Access model)

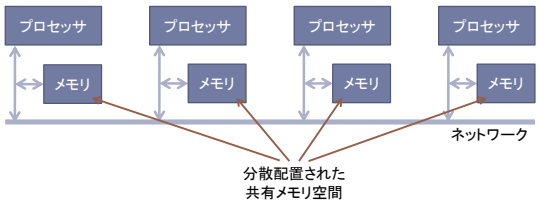
メモリ共有方法に基づく分類（2）

- UMA (Uniform Memory Access model)
  - すべてのプロセッサがメモリ空間を共有する．
  - すべてのプロセッサからのメモリアクセス速度が均一である．



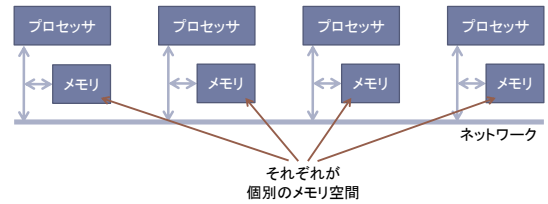
メモリ共有方法に基づく分類（3）

- NUMA　(Non Uniform Memory Access model)
  - すべてのプロセッサがメモリ空間を共有する.
  - プロセッサからのメモリアクセス速度が均一ではない.



メモリ共有方法に基づく分類（4）

- NORA　(NO Remote memory Access model)
  - すべてのプロセッサが独立したメモリを持ち，メモリ空間を共有しない.



結合の度合いに基づく分類（1）

- 結合の度合いに基づく分類
  - プロセッサ間の結合方法に着目した分類方法.
  - 以下のように大別される.
    - 密結合
    - 疎結合
    - クラスタ

結合の度合いに基づく分類（2）

- 密結合
  - プロセッサ間がバスを介して結合される方式.
  - 「メモリ共有方法に基づく分類」における“UMA”が相当する.
- 疎結合
  - プロセッサ間がネットワークを介して結合される方式.
  - 「メモリ共有方法に基づく分類」における“NUMA”，“NORA”が相当する.
- クラスタ
  - 独立した計算機群がネットワークにより結合される方式.
  - 疎結合の一種である.

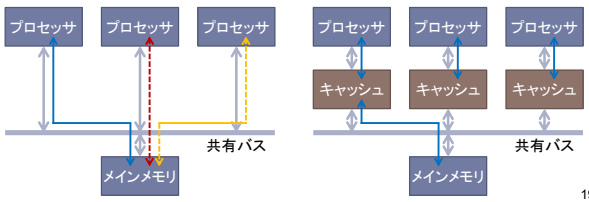
演習問題

- 問題1
  - 対象とする処理を，並列処理を用いて，高速化したい．単一のプロセッサに比べて4倍以上の高速化を実現するためには，何台以上のプロセッサを用いる必要があるか.
  - なお，対象とする処理は，その20%が並列化できない処理であるものとする．また，並列処理による速度向上比は，アムダールの法則に従っているものとする.

バス結合型並列アーキテクチャ

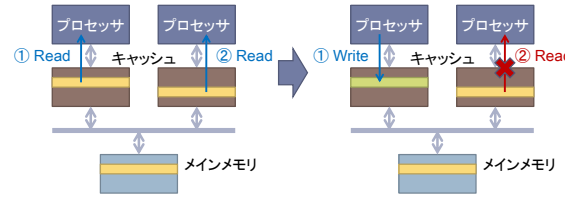
バス結合型並列アーキテクチャ

- バス結合型並列アーキテクチャ
  - 各プロセッサは，共有バスを経由して，共有メモリにアクセスする．
  - プロセッサ数を増やすと，共有バスの性能がボトルネックになる．
    そのため，各プロセッサにキャッシュを装備し，共有バスにおけるトラフィックを下げることにより，システム全体の性能低下を防止している．



バス結合型並列アーキテクチャにおける課題

- キャッシュの一致性問題（Cache Coherency：キャッシュコヒーレンシ）
  - キャッシュに対して書き込みが行われた場合，書き込みが行われたキャッシュとメインメモリとの間，書き込みが行われたキャッシュと他のキャッシュとの間に，データの不一致が生じてしまう．
  - そのため，メインメモリと複数のキャッシュにおける“内容の一貫性”を保持するための機構が必要になる．



キャッシュの一致性問題の解消方法
キャッシュとメインメモリ間のコヒーレンシ

- キャッシュとメインメモリ間のコヒーレンシを保つための方法
  - 書き込みが行われたキャッシュのブロックを，メインメモリに転送する．
  - 転送方法には，転送のタイミングに応じて，以下の2種類の方法がある．
    - ライト・スルー方式（write through）
      - キャッシュへの書き込みが生じた場合には，それと同時にメインメモリも更新する．
    - ライト・バック方式（write back）
      - キャッシュのブロックが追い出されることになったときに，メインメモリへの転送を行う．

キャッシュの一致性問題の解消方法
キャッシュ間のコヒーレンシ

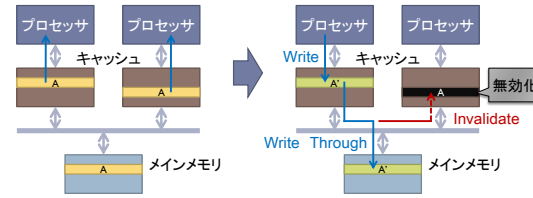
- キャッシュ間のコヒーレンシを保つための方法
  - あるキャッシュに書き込みが行われた場合には，その他のキャッシュに対して制御を行う．
  - 制御方法には，以下の2種類の方法がある．
    - ライト・インバリデート（write invalidate）
      - あるキャッシュに書き込みが行われた場合には，その他の当該キャッシュブロックを無効化する．
    - ライト・アップデート（write update）
      - あるキャッシュに書き込みが行われた場合には，その他の当該キャッシュブロックを即座に書き換える．

キャッシュの一致性問題の解消方法の分類

- キャッシュとメインメモリ間，キャッシュ間のコヒーレンシを保つための方法は，前述したライト・スルーとライト・バック，ライト・インバリデートとライト・アップデートの組み合わせにより，以下の4つに分類される．
  - ライト・スルー・インバリデート　（write through invalidate）
  - ライト・スルー・アップデート　（write through update）
  - ライト・バック・インバリデート　（write back invalidate）
  - ライト・バック・アップデート　（write back update）

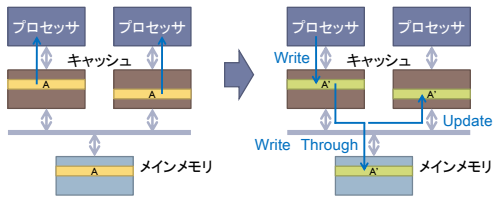
ライト・スルー・インバリデート

- あるキャッシュブロックに書き込みが行われた場合
  - 共有メモリに即座に書き込みを行う．
  - 書き込まれたキャッシュブロックを共有している他のキャッシュでは，該当するキャッシュブロックを即座に無効化する．



ライト・スルー・アップデート

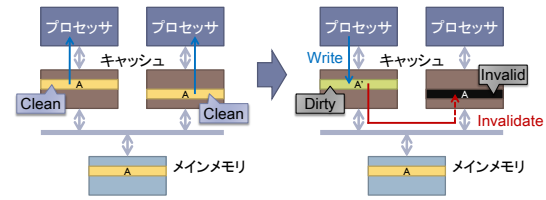
- あるキャッシュブロックに書き込みが行われた場合
  - 共有メモリに即座に書き込みを行う.
  - 書き込まれたキャッシュブロックを共有している他のキャッシュでは，該当するキャッシュブロックを更新する.



ここでは，基本プロトコルについて説明する.
他に，illinoisプロトコルやberkeleyプロトコルなどがある.

ライト・バック・インバリデート（1）

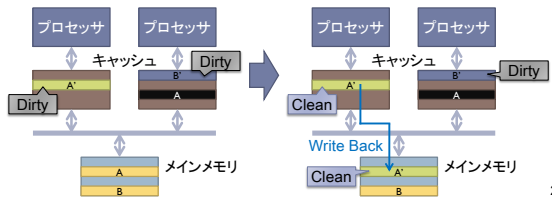
- あるキャッシュブロックに書き込みが行われた場合
  - 書き込まれたキャッシュブロックを共有している他のキャッシュでは，該当するキャッシュブロックを即座に無効化する.



ライト・バック・インバリデート（2）

- キャッシュからの読み出しがミスヒットした場合（その1）
  - 置換対象ブロック（メインメモリに追い出すブロック）が“dirty”の場合には，当該ブロックを共有メモリに書き込む.

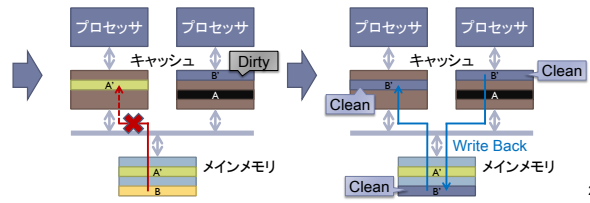
左側のキャッシュにおいて，ブロックAを追い出して，ブロックBを読み込む場合



ライト・バック・インバリデート（3）

- キャッシュからの読み出しがミスヒットした場合（その2）
  - メインメモリから読み出すブロックのコピーが存在しており，かつそれが“dirty”の場合には，その“dirty”なブロックを，共有メモリに書き込む.
    その後，共有メモリからキャッシュへ，所望のブロックを書き込む.

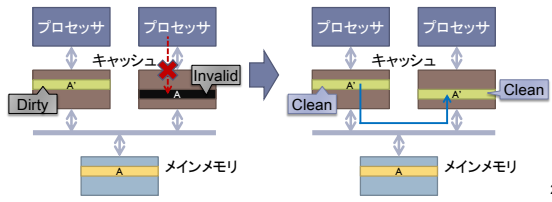
左側のキャッシュにおいて，ブロックAを追い出して，ブロックBを読み込む場合



ライト・バック・インバリデート（4）

- キャッシュへの書き込みが無効化状態のためミスヒットした場合（その1）
  - 当該ブロックの“dirty”なコピーが存在している場合には，そのブロックを，キャッシュへ書き込む.

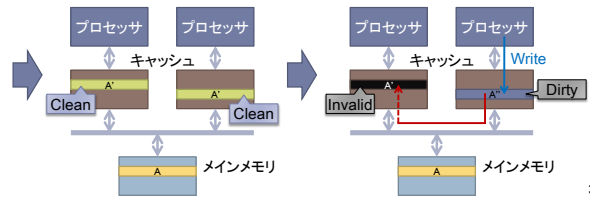
右側のキャッシュにおいて，ブロックAに書き込む場合



ライト・バック・インバリデート（5）

- キャッシュへの書き込みが無効化状態のためミスヒットした場合（その2）
  - キャッシュに書き込みを行い，書き込まれたキャッシュブロックを共有している他のキャッシュでは，該当するキャッシュブロックを即座に無効化する.

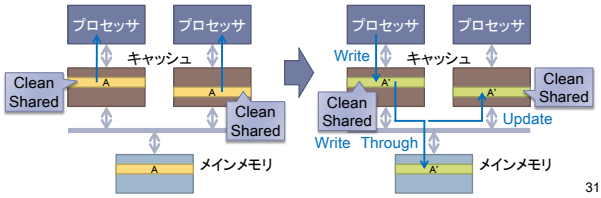
右側のキャッシュにおいて，ブロックAに書き込む場合



ここでは，Fireflyプロトコルについて説明する．
他に，Dragonプロトコルなどがある．

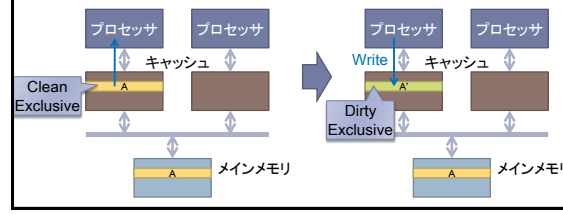
ライト・バック・アップデート（1）

- 共有されているキャッシュブロックに書き込みが行われた場合
  - 共有メモリに即座に書き込みを行う．（すなわち，ライト・スルー．）
  - 書き込まれたキャッシュブロックを共有している他のキャッシュでは，該当するキャッシュブロックを更新する．



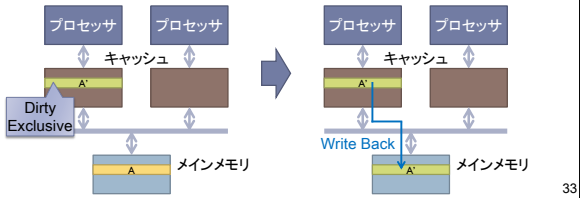
ライト・バック・アップデート（2）

- 共有されていないキャッシュブロックに書き込みが行われた場合（その1）
  - 書き込んだブロックを“Dirty Exclusive”とする．



ライト・バック・アップデート（3）

- 共有されていないキャッシュブロックに書き込みが行われた場合（その2）
  - 上記ブロックが追い出される場合には，当該ブロックを共有メモリに書き込む．



ライト・スルー型とライト・バック型の比較

- ライト・スルー型
  - ライト・スルー・インバリデート，ライト・スルー・アップデートとも，制御が比較的に簡単である．
  - 共有メモリへの書き込みのたびにバスを利用するので，効率が良くない．
- ライト・バック型
  - ライト・スルー型に比べて，制御が複雑である．
  - ライト・スルー型に比べて，共有メモリへのアクセス頻度は少なく，効率が良い．

一般に，ライト・バック型が用いられている．

演習問題

- 問題2
  - キャッシュとメインメモリ間，キャッシュ間のコヒーレンシを保つための方法に，以下のような方法がある．それぞれについて，概略説明せよ．
    - ライト・スルー・インバリデート　（write through invalidate）
    - ライト・スルー・アップデート　（write through update）

演習問題

- 問題3
  - ライト・バック・インバリデートの基本プロトコルを用いているものとして，以下の問いに答えよ．
  - プロセッサAのキャッシュとプロセッサBのキャッシュに，同一のキャッシュブロックが存在しており，いずれも Clean な状態にあるものとする．
  - いま，プロセッサAが，上記キャッシュブロックに書き込みを行ったものとする．このとき，
    1. プロセッサAの当該キャッシュブロックは，Clean・Dirty・Invalid のどの状態になるか．
    2. プロセッサBの当該キャッシュブロックは，Clean・Dirty・Invalid のどの状態になるか．

演習問題

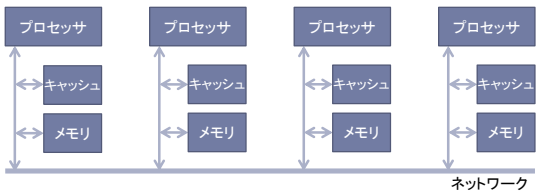
- 問題4
  - ライト・バック・アップデートのFireflyプロトコルを用いているものとして，以下の問いに答えよ．

1. プロセッサAが，Clean Exclusive なキャッシュブロックに対して，書き込みを行ったものとする．このとき，プロセッサAの当該キャッシュブロックは，Clean Shared・Clean Exclusive・Dirty Exclusive のどの状態になるか．
2. プロセッサAが，Clean Shared なキャッシュブロックに対して，書き込みを行ったものとする．このとき，プロセッサAの当該キャッシュブロックは，Clean Shared・Clean Exclusive・Dirty Exclusive のどの状態になるか．

ネットワーク結合型並列アーキテクチャ

ネットワーク結合型並列アーキテクチャ

- ネットワーク結合型並列アーキテクチャ
  - 各プロセッサは，ネットワークを介して結合される．
  - 単一バスに接続できるプロセッサの数には，実装上の問題により，限度がある．これに対して，ネットワーク結合型では，より多くのプロセッサを接続することができる．



クラスタ

クラスタ

- クラスタ
  - 独立した計算機群を，ネットワークにより結合させたもの．
- クラスタの特徴
  - 長所
    - 計算能力が同等な大型マシンに比べ，安価に構成することができる．
    - システムを停止させずに，マシンを取り換えることが比較的容易である．
  - 短所
    - ネットワーク結合による通信は，バス結合による通信よりも遅い．
    - メインメモリが分割されているため，大容量のメインメモリを利用できない．
    - 管理コストが高い．
      （複数のマシンを個別に管理するのと同程度のコストがかかる．）