Johnny Cache: the End of DRAM Cache Conflicts (in Tiered Main Memory Systems)

2025/6/12

オペレーティングシステム特論

コンピュータ科学専攻 修士2年

穴田 穂乃香

概要

B. Lepers and W. Zwaenepoel, "**Johnny Cache: the End of DRAM Cache Conflicts (in Tiered Main Memory Systems)**", in *Proc. OSDI*, 2023.

- DRAMより下層に低速で大容量なメモリ (e.g. 永続メモリ (PMEM)) が存在する **tiered memory system** において、DRAMをキャッシュメモリとして扱う手法を研究
- ページ割り当てを工夫することでDRAMのcache conflictを減らし、
  従来のソフトウェアベースのtiered memory systemより高い性能を実現した

発表内容

- 背景と課題
- 提案手法の設計と実装
- 評価方法と結果

発表内容

- **背景と課題**
- 提案手法の設計と実装
- 評価方法と結果

Tiered Memory System

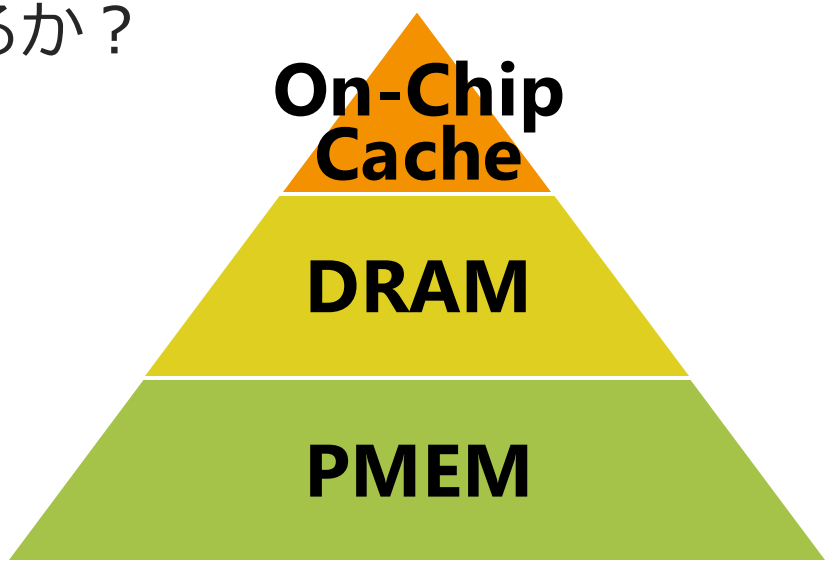
DRAMよりも下層に低速で大容量なメモリが存在するメモリシステム

例 : SSD, PMEM (Intel Optane etc.), CXL memory etc.

注 : 本研究では主にPMEMを想定しているため、以下PMEMとして扱う

どのようにDRAMとPMEMに置くデータを制御するか？

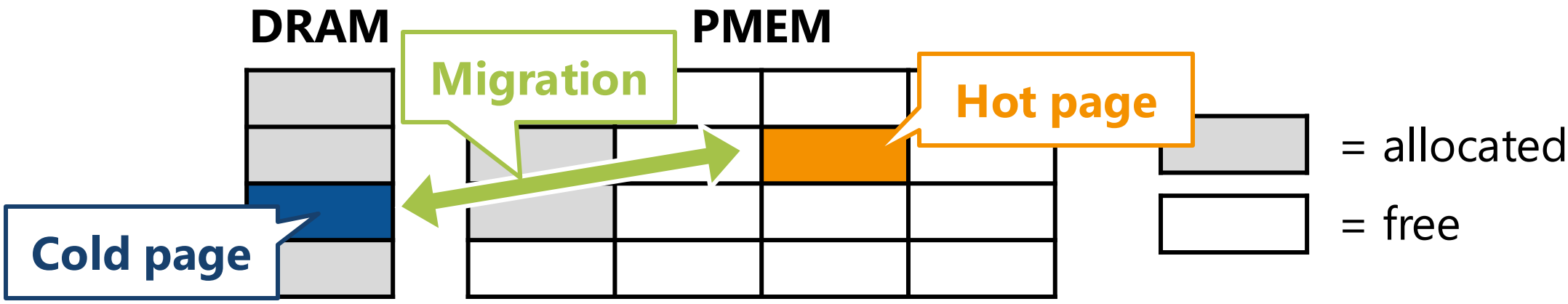
1. **ソフトウェア制御**
   OSが各ページをDRAMとPMEMの
   どちらに割り当てるか決定
2. **ハードウェア制御**
   DRAMをキャッシュメモリとして扱う



ソフトウェア制御

OSがDRAMとPMEM双方を一続きの物理アドレス空間として扱う

- ナイーブな割り当てではPMEM側にアクセス頻度の高いデータが集中する可能性 ☹
- デーモンプロセスが各ページのアクセス頻度を監視
  → **頻繁にアクセスされているページ (hot page) をDRAMに移動 (migration)**



ソフトウェア制御の限界

ソフトウェア制御にはいくつかの課題がある：

1. **ページ移動のオーバーヘッドが大きい**
   - ページ単位の移動 (4KB or 2MB) → データ読み書き量が大きい
   - ページテーブルの書き換え & TLBフラッシュも必要
2. **DRAMに置くデータをページ粒度でしか制御できない**
   → 1つのページにhot dataとcold dataが混ざっていると効率が悪い
3. **ページ移動はデーモンによって非同期的 & 遅延して行われる**
   → Hot pageであっても移動まではPMEMに毎回アクセスする必要
4. **デーモンによってメモリアクセスをサンプリングするオーバーヘッド**

ハードウェア制御

DRAMもキャッシュ階層の一部として扱う (= DRAM cache)

- OSから見える物理アドレス空間 = PMEM
- 1-way (direct-mapped) cache とするのが一般的 (e.g. Intel Optane)
  - On-chip cache より大容量な DRAM cache では最も効率的 [1]

[1] M. K. Qureshi and G. H. Loh. “Fundamental Latency Trade-Off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design”, In *Proc. MICRO*, 2012.

ハードウェア制御の利点

ソフトウェア制御の限界		ハードウェア制御ではこれらを克服：
ページ移動のオーバーヘッド (ページテーブル書き換え etc.)	→	ページテーブル変更や TLBフラッシュは不要
DRAMに置くデータを ページ粒度でしか制御できない	→	キャッシュライン粒度 (= 64B程度) で制御
ページ移動が非同期的	→	Hot dataは初回アクセス時に 同期的にDRAMに移動
メモリアクセスをサンプリングする オーバーヘッド	→	不要

現状のハードウェア制御の課題

- ソフトウェア制御では起こらない **cache conflict** が発生する
- ソフトウェア観点の情報を利用できない (e.g. データのアクセス頻度)

アクセス頻度が高い2つのデータが同じキャッシュラインに対応すると
頻繁な conflict miss によってパフォーマンスが低下

- 通常のSRAM+DRAMに比べてDRAM+PMEMは低速
  → キャッシュミス時のオーバーヘッドも大きい ☹

先行研究や実システムの事例

- **Intel Optane**では2つのモードを提供
  - **App-direct mode** → ソフトウェア制御
  - **Memory mode** → ハードウェア制御
- **先行研究ではソフトウェア制御の工夫について数多く研究**
  → それらはハードウェア制御の性能を上回っている
  - **HeMem** [2] : SOTA; migrationによるオーバーヘッド削減に着目
  - 過去にも Thermostat [3], Nimble [4] などいくつか提案

[2] A. Raybuck et al. “HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM”, in *Proc. SOSP*, 2021.
[3] N. Agarwal and T. F. Wenisch. “Thermostat: Application-transparent Page Management for Two-tiered Main Memory”, in *Proc. ASPLOS*, 2017.
[4] Z. Yan et al. “Nimble Page Management for Tiered Memory Systems”, in *Proc. ASPLOS*, 2019.

本研究の貢献

Tiered memory systemのハードウェア制御はソフトウェア制御に比べて
利点も多いが、
ソフトウェアレベルの情報を使えないこと & cache conflictのオーバーヘッドが
原因でパフォーマンスが不十分 ☹

Cache conflictを減らすような物理ページ割り当てをOSカーネルに実装することで、
ハードウェア制御の欠点を克服し、既存手法より高いパフォーマンスを実現 ☺

発表内容

- 背景と課題
- **提案手法の設計と実装**
- 評価方法と結果

提案手法 : Johnny Cache

目標 : ページ割り当ての工夫によりページ間でのcache conflictを最小化

提案1 : 静的ポリシー (JC-static)

- 物理ページの割り当て状況のみから (conflictが起きないように)
  ページ割り当てを決定

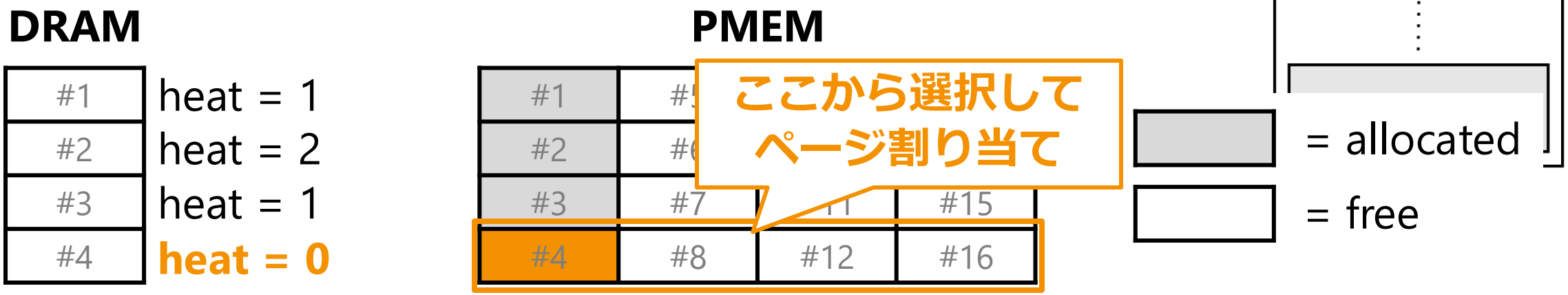
提案2 : 動的ポリシー (JC-dyn)

- メモリアクセスのサンプリング結果に基づいてページ割り当てを決定
- 頻繁にconflictする物理ページを他の場所に移動

共通の設計

- **bin** = DRAMキャッシュ上で1ページに対応するキャッシュラインの集合
- カーネルは以下のメタデータを保有 :
  - 各物理ページの割り当て状況
  - 各bin (と動的ポリシーの場合は各ページ) の **heat**

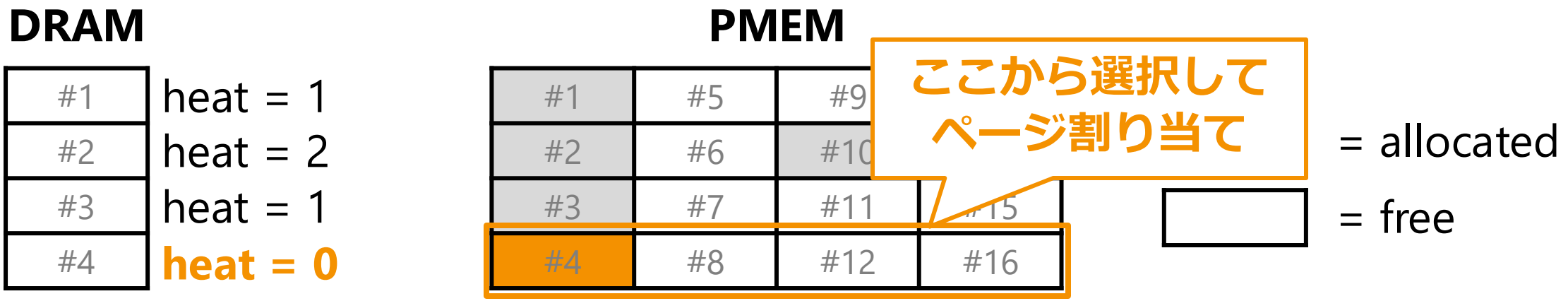
  ※ 128 GB DRAM + 1 TB PMEM であれば合計で 50 MB 以下
- **heatが小さいbinに対応する物理ページを割り当て**



提案1：静的ポリシー

heat = そのbinに対応する割り当て物理ページ数

- ページ割り当て / 解放時にheatを増減すればOK → オーバーヘッド最小 ☺



提案2：動的ポリシー (ページ割り当て)

デーモンによってメモリアクセスをサンプリング

→ **アクセス頻度に基づいてheatを計算する**

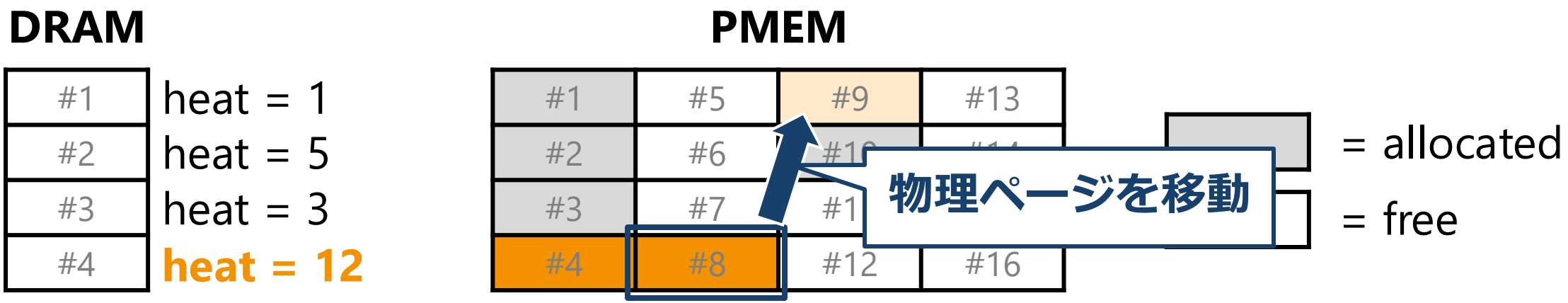
- binだけでなく物理ページにもheatを保持
- メモリアクセスを観測 → 対応する物理ページとbinのheatを増やす
- 物理ページのheatが非常に大きくなったらcooling

→ アクセス頻度が小さいbinに物理ページを割り当て

提案2：動的ポリシー (動的ページ移動)

- Migration daemon が定期的にheatが大きいbinに対応する物理ページを調査
- そのbinに**2つのhot pagesが割り当て→ 片方を他の場所にremap**

※ Hot page = heatが閾値より大きいページ



実装

- Intel Optane の memory mode を対象として、
  Linuxカーネルに対するパッチとして実装
  - 1-way cache 向けの実装だが、少しの変更でset-associativeにも対応可能
- ページ割り当て時 (= page fault handler)
  → **ポリシーに従ったページ割り当てを実装**
- ページ割り当て (= page fault handler) 時と
  ページ解放 (= page unmap handler) 時に**物理ページ割り当て情報を更新**

動的ポリシーの実装

メモリアクセスをサンプリングする必要

→ Intel CPU の **Processor Event-based Sampling (PEBS)** を利用

- 特定のイベント (e.g. PMEMからのload) カウンタが閾値を超えたら、
  そのイベントの詳細 (物理アドレスなど) を専用のメモリ領域に記録
  (= N回に1回のイベントが記録される; N=5000)
- **PEBSのメモリ領域をスレッドが監視し、更新されたら以下の処理を実行**
  1. アクセスされた物理ページと、対応するbinのheatをincrement
  2. 更新したbinがmigrationの条件を満たすようになったら、
     migration daemonを起こしてmigrationを実行

発表内容

- 背景と課題
- 提案手法の設計と実装
- **評価方法と結果**

評価環境

- **48GB DRAM cache + 512GB PMEM** (Intel Optane NV-DIMMs) が利用可能な単一NUMAノード上で実験
  ※ Intel Xeon Gold 6230 core x 40 + 128GB DRAM + 8x128GB Intel Optane NV-DIMMs 搭載のtwo-node NUMA machineのうちの1ノードを使用
- 以下の4つを各ワークロード (後述) について評価:
  - Memory mode + **素のLinux**
  - App-direct mode + **HeMem** (既存のソフトウェアベース手法)
  - Memory mode + **JC-static**
  - Memory mode + **JC-dyn**

評価したワークロード

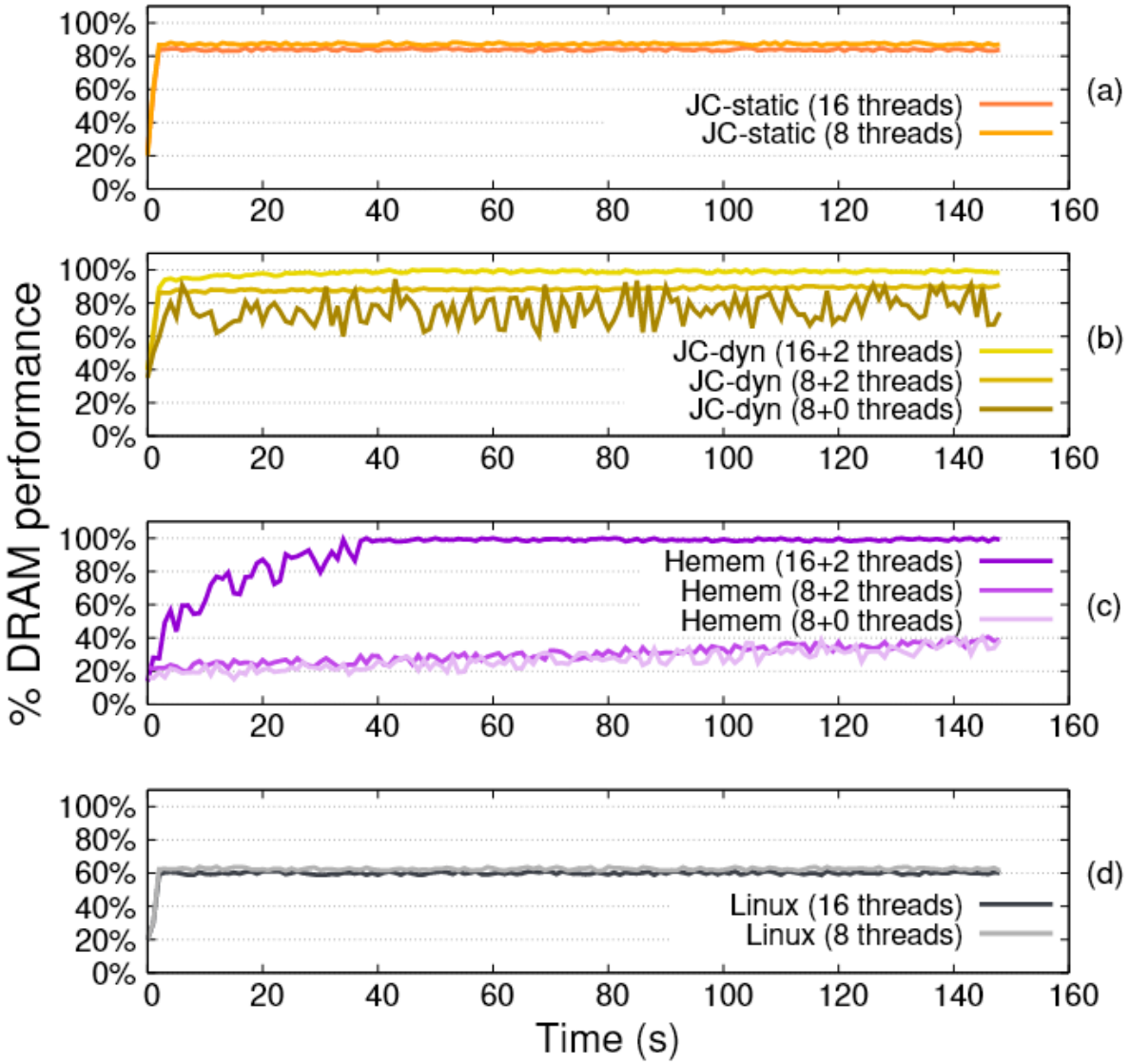
原則として先行研究のHeMemに準拠 + 一部追加

1. **GUPS** : 大規模配列をスレッド並列で更新
   - 配列の一部 (hot region) に対して90%のアクセスが集中
     → DRAMに配置するデータが重要
2. **BC** : べき乗則グラフにおける媒介中心性の計算
3. **Silo** : メモリ内データベース
4. **Masstree** : メモリ内 key-value store
5. **NAS benchmark suite**
   (BT.D, CG.D, EP.D, LU.D, MG.D, SP.D, UA.D, CG.E, EP.E, MG.E)

評価結果：GUPS #1 (96GB, hot data=9.6GB)

- JC-staticやLinuxと違い、JC-dynやHeMemは最大パフォーマンス到達に時間を要する
- JC-dynの方がHeMemより早く最大パフォーマンスに到達
- Linuxは全体としてパフォーマンスが低い
- スレッド数が少ない場合にはHeMemはパフォーマンスが非常に低い

※ 100% = すべてのhot dataをDRAMに置いた場合



考察 (GUPS #1) 1/2

- **JC-staticやLinuxと違い、JC-dynやHeMemは最大パフォーマンス到達に時間を要する**
  - 物理ページのアクセス頻度を観測してmigrationを完了する必要があるため
- **JC-dynの方がHeMemより早く最大パフォーマンスに到達**
  - HeMemはmigration完了まではパフォーマンスが非常に低い
  - HeMemはJC-dynよりmigrationの量が多い
    (JC-dynはconflictしたページのみ移せばいいため)

  → **Johnny Cache は warm up なしで高パフォーマンスを実現 ☺**

考察 (GUPS #1) 2/2

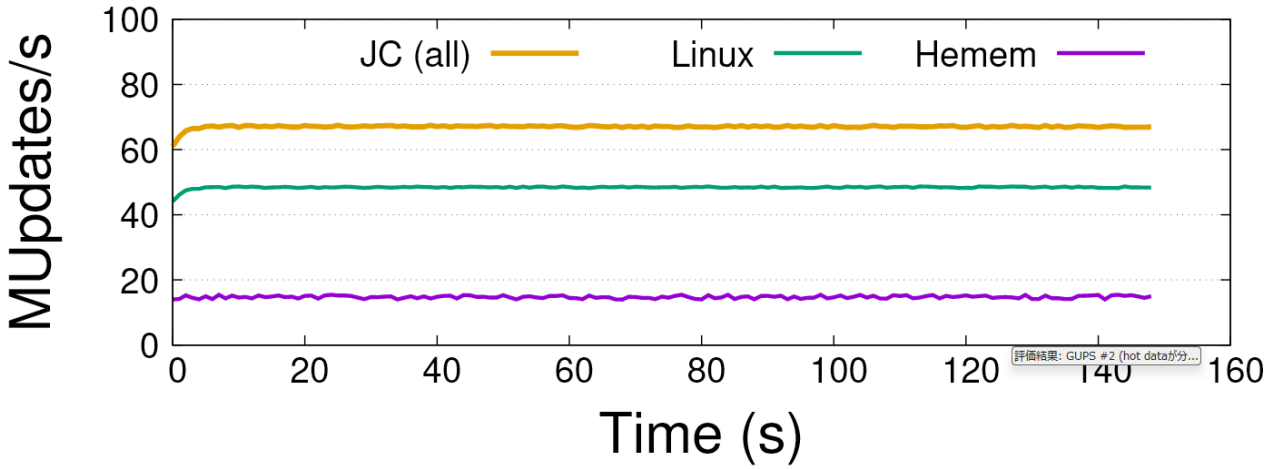
- **Linuxは全体としてパフォーマンスが低い**
  - ランダムな割り当てではcache conflictが多発している
- **スレッド数が少ない場合にはHeMemはパフォーマンスが非常に低い**
  - Coolingの頻度に比べてアクセス頻度 (=heatの上昇速度) が低いのが原因 ☹

    ※ cooling = heatが極端に大きくなったページのheatを下げる
  - Cooling頻度を全てのアプリケーションに最適にすることはできない

    → **そのようなfine-tuningが不要なのがJohnny cache の強み** ☺

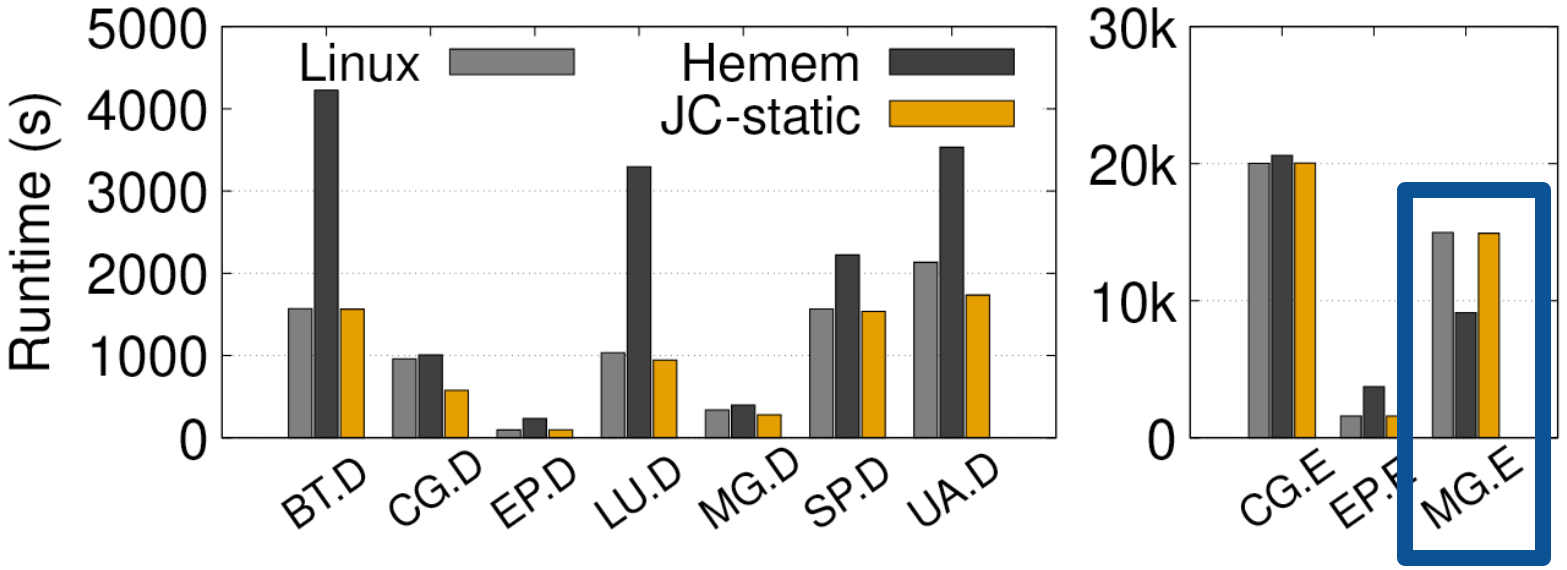
評価結果: GUPS #2 (Hot valuesが分散)

- Hot values の空間局所性が低いときはHeMemのパフォーマンスが低い
  → Hot valueを含むすべての物理ページをDRAMに置くことは不可能なため
  - 同様にJC-dynはJC-staticと変わらないパフォーマンスしか達成できず
- しかし、**JC (やLinux) はライン単位 (64B) でDRAMにデータを置ける**
  → HeMemよりは高いパフォーマンスを達成



提案手法がうまくいかないケース

- NAS Benchmark suite のうち MG.E では HeMemより低いパフォーマンス
  - Uniform access でデータの再利用がないため、
    キャッシュではなく初めから一部のデータがDRAMにあるHeMemが有利



まとめ

- Tiered memory system においてDRAMをキャッシュとして使う方式は、ソフトウェアレベルの情報を使えないこと & cache conflictのオーバーヘッドが原因でパフォーマンスが不十分だった
- 本研究では、Cache conflictを減らすような物理ページ割り当てやページ移動をOSカーネルに実装する**Johnny cache**によって、これらの課題を克服した
- 実験を行い、従来の研究で提案されてきたソフトウェアベースの手法を上回るパフォーマンスを達成した