Transparent HugePage Support (THP)

CHAN Eros
CHEN siyuan
ZHOU runlin

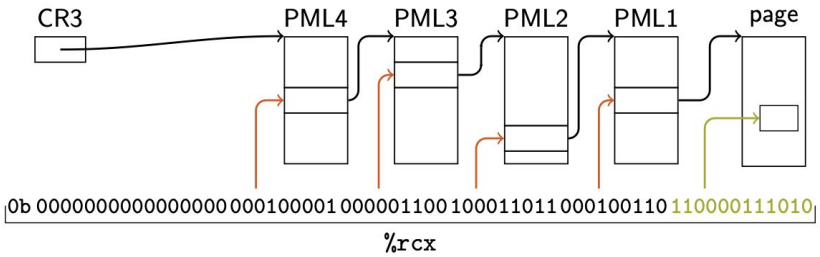
Contexte et concepts de base

Page

- Utilisée pour la mémoire virtuelle
- Bloc mémoire de taille fixe
  - Taille de 4 Ko
- Ce qu’elle permet :
  - Isolation des adresses entre les différents processus
  - S’émanciper des contraintes de la mémoire contigüe
  - Évite la fragmentation externe

Table des pages

- Structure de données arborescente
  - Pour chaque adresse virtuelle, la table des pages fournit l'adresse physique correspondante en RAM
- Utilisée par la MMU
- Décomposée en 4 niveaux
  - Un lookup implique 4 accès mémoire RAM par requête
- Pour éviter ce (sur)coût, on utilise un cache matériel



Translation Lookaside Buffer

- Mémoire cache interne à la MMU
- Stocke temporairement des paires (adresse virtuelle, adresse physique)
- Taille limitée
  - Ne contient qu'un nombre restreint d'entrées
- Améliore les performances
  - Réduit le nombre d'accès à la table des pages

- Nouvelle entrée à chaque traduction réussie
- Vidé lorsque le CR3 est modifié

Translation Lookaside Buffer

- Un lookup dans la table des pages est lent
  - Stockée en RAM (~ 100 ns)
  - Présence de plusieurs niveaux d’indirections
- Le TLB évite ces accès coûteux en RAM

- TLB hit = accès immédiat
  - 1 ns / ~ 5 cycles CPU
- TLB miss = absence de l’adresse en cache, qui implique un lookup classique avec plusieurs accès mémoire en RAM
  - 400 ns / ~ 500-1000 cycles CPU

Translation Lookaside Buffer : Limites

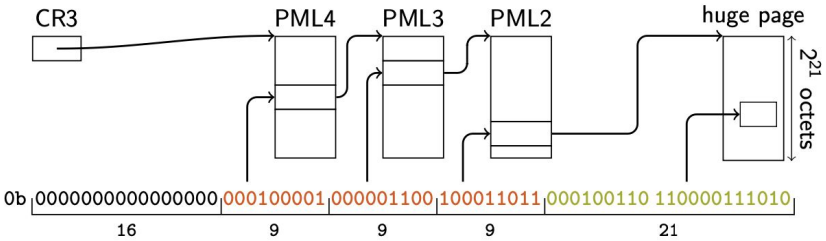
- Capacitée limitée
  - Nombre restreint d’entrées, conduisant à des TLB misses
- Impact sur les performances en présence de TLB misses
  - Consommation excessive de cycle CPU
- Surcharge dû à l’invalidation
  - La modification de la table des pages invalide le TLB
- Phénomène accentué par la multiplication des entrées de la table des pages
  - Une page trop petite e.g. de 4 Ko entraîne plus d’entrées sur la table des pages

Translation Lookaside Buffer : Solutions

- Augmenter la taille du TLB
  - Approche limitée par des contraintes matérielles
- Prefetching des entrées du TLB
  - Anticiper les accès mémoire pour limiter les TLB misses
    - Exemple : Speculative TLB Prefetching
- Optimisations logicielles possibles
  - Augmenter la taille des buffers pour limiter la fragmentation et diminuer la pression sur le TLB
    - Exemple : jemalloc, tcmalloc…
- **Huge Page**

Huge Page

- Page de taille de 2 Mo (512 x 4 Ko)
  - La taille doit couvrir une plage alignée de mémoire (4 Ko, 2 Mo, 1 Go…)
- Réduit les TLB misses
  - Si les pages sont plus grandes, il y a moins d'entrées dans la table des pages
- Réduction de la fragmentation



Huge Page : Configuration “manuelle”

- Réserver 512 pages de 2 Mo
  - `echo 512 > /proc/sys/vm/nr_hugepages`
- Montage du Pseudo filesystem spécifique aux Huge Pages
  - `mount -t hugetlbfs none /dev/hugepages`

- Avantage
  - Contrôle “fin”
- Inconvénients
  - Absence d’une stratégie dynamique, contrainte de réservation à l’avance
  - Nécessite de recompiler les applications pour en profiter

Huge Page : Configuration “automatique”

- Le noyau détecte les zones mémoires “chaudes” et les regroupe en Huge Page
- S’active avec depuis un fichier de configuration
  - `echo always > /sys/kernel/mm/transparent_hugepage/enabled`

- Avantage
  - Transparent, le kernel s’occupe de tout
- Inconvénients
  - Des latences peuvent apparaître en cas de défragmentation de la mémoire
  - Performances dégradées pour certains pattern d’accès mémoire

Transparent Huge Page

- Fonctionnalité du noyau qui gère “automatiquement” l’allocation des Huge Pages
- Profite de la localité lors de nombreux accès dans un même bloc de mémoire
- Réduit le nombre de fautes de page
    - Pour une page de 2 Mo, 1 accès mémoire = une seul faute contre 512 pour 4 Ko

- Allocation automatique repose sur deux mécanismes
    - Direct THP allocation
    - khugepaged
- Activée par défaut sur la plus pas des systèmes

Transparent Huge Page : Direct THP allocation

- Quand un processus alloue de la mémoire via `mmap()` ou `brk()`, le noyau tente d'allouer directement des Huge Pages

- Si la taille demandée est au moins de 2 Mo
- Si l'adresse mémoire est alignée sur 2 Mo
- Si la mémoire physique n'est pas fragmentée
  - Si c'est le cas, attribution de page de 4 Ko
  - Le kernel essayera plus tard de "collapser" en Huge Page via khugepaged

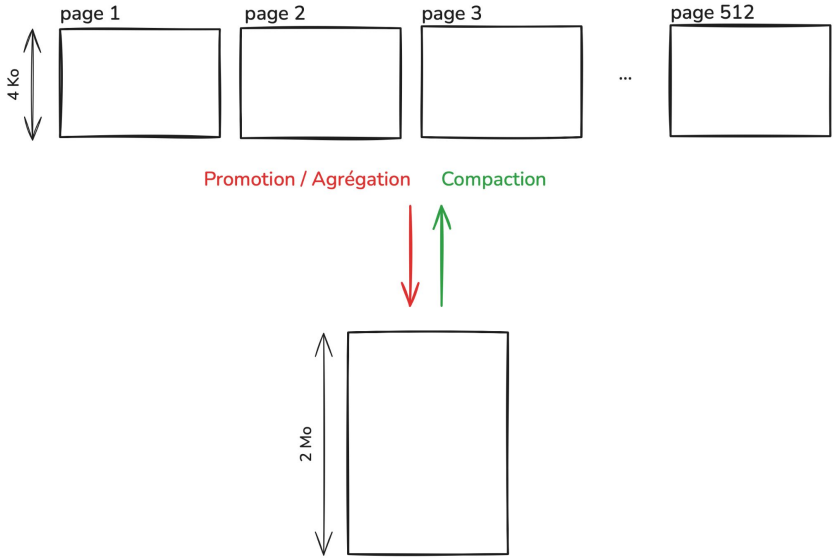
Transparent Huge Page : khugepaged

- Démon interne au noyau Linux
- Parcour périodiquement la mémoire pour
  - Détecter les séquence de pages de 4 Ko qui pourraient être fusionnées

- Promotion / Aggrégation
  - En présence d’ensemble contigu
  - Décide s’il faut ou non aggréger une zone mémoire en Huge Page
- Compaction
  - Décide s’il faut ou non défragmenter la mémoire

Transparent Huge Page : khugepaged

- Démon interne au noyau Linux qui parcours périodiquement la mémoire pour détecter les séquence de pages de 4 Ko fusionnables
- Promotion / Agrégation
  - Décide s'il faut ou non aggréger une zone mémoire en Huge Page
- Compaction
  - Décide s'il faut ou non défragmenter la mémoire



Mécanismes et Fonctionnement Interne

Limitation du THP dans Linux

- Fragmentation de la mémoire
- Cout de fractionnement et de fusion
  - Migration des pages
  - Échec de l'optimisation de la TLB
- Flexibilité insuffisante

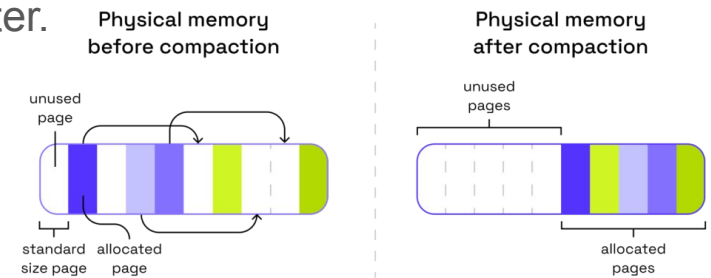
Global THP Controls : always, madvise et never

- Le noyau Linux fournit une interface pour configurer dynamiquement THP avec sysfs
  - `echo <option> > /sys/kernel/mm/transparent_hugepage/enable`

- Les options :
  - `always` : le système fusionne automatiquement toutes les régions de mémoire en Huge Page chaque fois que cela est possible
  - `madvise` : seules les régions de mémoire marquées par madvise (MADV_HUGEPAGE) sont converties en Huge Page
  - `never` : désactive la fonctionnalité

Global THP Controls : always, madvise, defer et never

- Possible de limiter les efforts de défragmentation de la mémoire pour générer des pages anonymes sous **/sys/kernel/mm/transparent_hugepage/defrag**
  - **always :** directly reclaim pages and compact memory in an effort to allocate a THP immediately.
  - **madvise (defaut):** will enter direct reclaim like always but only for regions that are have used madvise (MADV_HUGEPAGE)
  - **defer :**
    - wake ***kswapd*** in the background to reclaim pages
    - wake ***kcompactd*** to compact memory so that THP is available
    - khugepaged then install the THP pages later.
  - **defer + madvise**
  - **never :** should be self-explanatory.

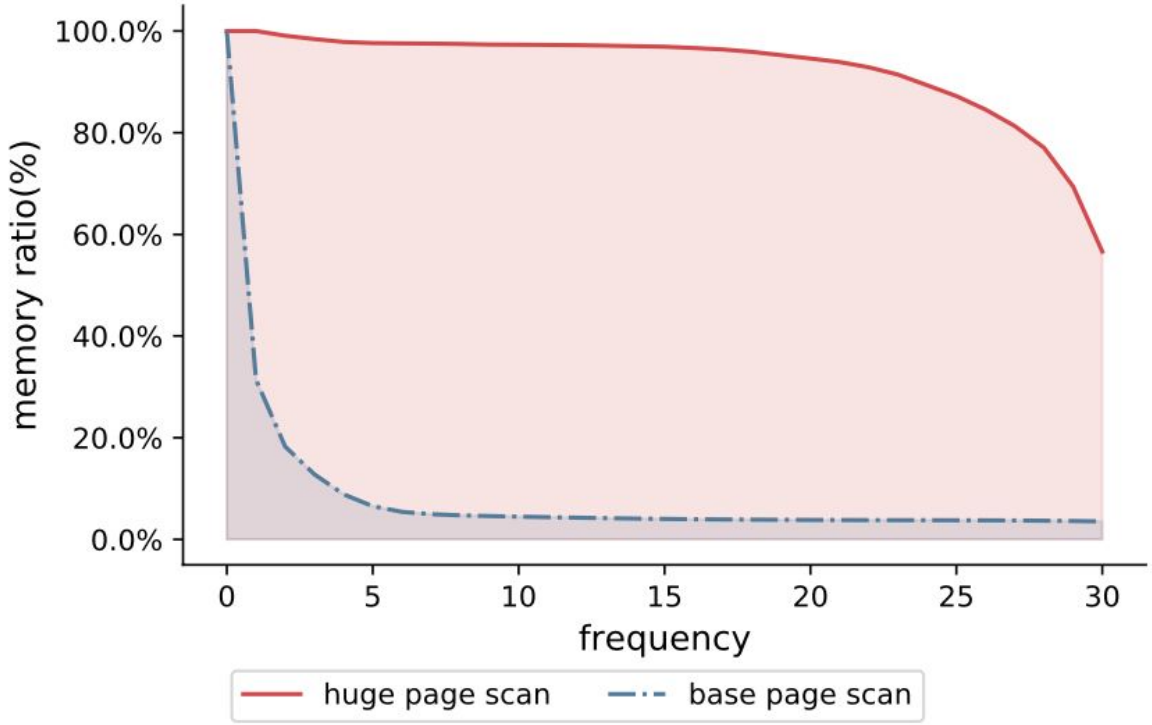


How compaction conceptuallly works

FHPM (Fine-grained Huge Page Management)

- En besoin de traduire entre GVA et GPA (HPA) dans les VMs
  - besoin en mémoire élevé -> fréquent absences de la TLB.
  - THP peut améliorer l'efficacité de la traduction d'adresses
- Une fois qu'une carte de Huge Page est créée, toutes les pages de base partagent la même entrée dans Extended Page Table (EPT)
- L'hyperviseur ne peut pas obtenir d'informations d'accès détaillées lorsqu'il utilise les bits A/D
  - Causé HOT bload
  - Influence Page Sharing

FHPM : Hot Blot

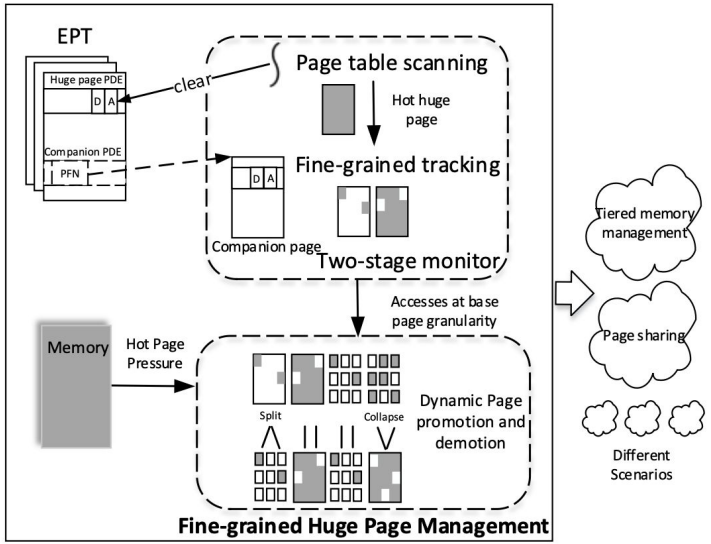


FHPM : Principes de conception et d'optimisation

- Les interfaces traditionnelles n'opèrent que sur la table de pages interne de la VM et ne mettent pas immédiatement à jour l'EPT
  - Déclenche un grand nombre de VM-exit
- Dans FHPM :
  - Remplissage proactif de l'EPT : m-à-j automatique après une opération de division ou de fusion
  - Fractionnement et fusion de pages adaptés aux VM

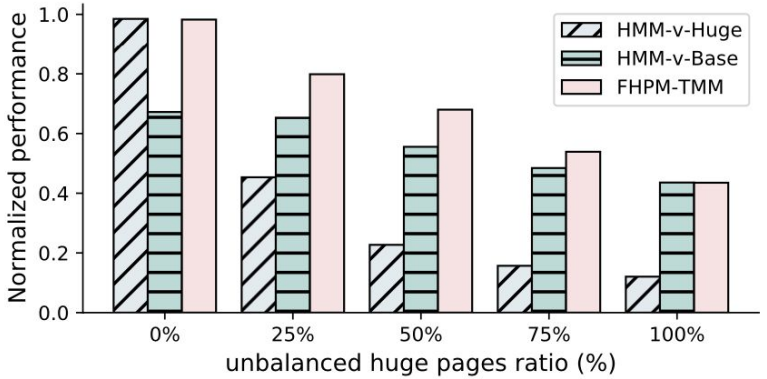
FHPM : Principes de conception et d'optimisation

- Surveillance des accès
  - Surveillance à grande échelle : classer la Huge Page en deux catégories : “chaude” et “froide”
  - Surveillance fine : redirige une PDE (Page Directory Entry) vers une page compagnon
- Mécanisme de redirection vers une Companion Page :
  - Comme une EPT temporaire
  - Dispose d'un bit A/D indépendant
  - Lorsque la surveillance terminée, le système rétablit les pages d'accompagnement dans le mappage original



FHPM : Principes de conception et d'optimisation

- Le Page Skew Ratio (PSR) mesure si les accès aux pages sont équilibrés
  - Un PSR élevé indique que certaines pages sont fréquemment accédées
  - Un PSR faible indique que les accès aux pages sont équilibrés
- Démotion : Diviser une page pas équilibrée
- Promotion : Fusionner les données chaudes



TLB et migration des pages

- Approche Linux classique
  - Division des THP en pages de 4 Ko lors de la migration
  - Perte des avantages du TLB
  - Surcharge due au traitement de nombreuses petites pages

- **Migration for Multi-Tiered Large Memory Systems (MTM)**
  - Migration directe des THP sans division

MTM : Stratégie “THP-aware”

- Comment on évite la fragmentation ?
  - THP-aware Memory Region Management
  - Fast Promotion & Slow Demotion

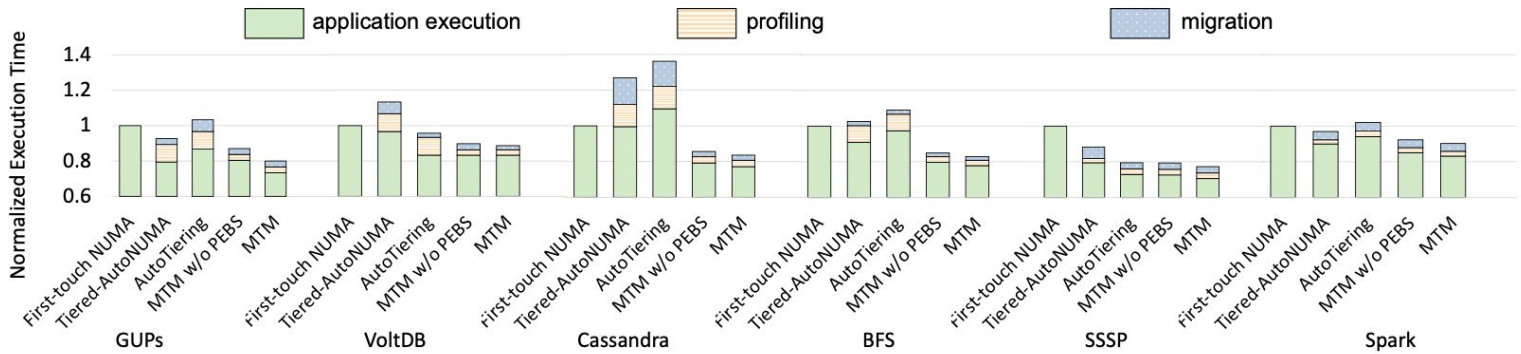
MTM : THP-aware Memory Region Management

- Mécanismes :
  - Reconnaissance et migration directe des Huge Page en utilisant ***move_memory_regions()***
  - Préservation de l'alignement lors des opérations de fusion et de division
  - Execute la migration de page vers les données chaudes et froides

MTM : Fast Promotion and Slow Demotion

- Vérifie si la page sélectionnée est une Huge Page
- Échantillonnage global de la Huge Page pour capturer les accès
  - En utilisant Performance Counter et EPT Scan pour identifiées les données chaudes
  - Échantillonnage à plusieurs niveaux
- Amélioration de la qualité du profilage
  - Les données chaudes sont directement transférées vers la mémoire rapide
  - Expulsion des données froides couche par couche
  - Multi-View Tiered Memory : choisir le thread plus fréquemment accédées pour chaque nœude du NUMA

MTM : Analyse et échantillonnage



Cas d’usage et benchmarks

Automated Reasoning

- Algorithme Conflit Driven Clause Learning (CDCL)
  - Exemple d’algorithme: SAT, SMT solver…
- Problèmes :
- Durée de calcul longue
- Importante ressource computation
- Nombreux accès mémoire non séquentiels
  - Engendre de nombreux TLB misses

Automated Reasoning : Solution

- Application d’un patch sur la glibc
  - Édition de liens nécessaires vers la nouvelle glibc
- Rend disponible une nouvelle option `GLIBC_THP_ALWAYS=1`
  - Force l’activation de la THP pour les allocations mémoire des solvers

Automated Reasoning : Benchmark

Résultat:

1. 10% moins de temps d'exécution
2. TLB miss largement amélioré, sur certain benchmark 1% comparer au précédent

solver	#	t_n	t_{thp}	s	TLB_n	TLB_{thp}	r_{tlb}
glucose	189	4.58	3.72	18.70	2.60E+11	6.71E+09	2.59
lingeling	177	6.18	5.91	4.76	4.93E+10	5.13E+08	1.04
winner19	194	7.46	6.24	16.67	3.29E+11	1.35E+10	4.10
mergesat	176	7.31	6.22	14.53	3.02E+11	1.36E+10	4.50
minisat	170	7.26	6.09	15.97	2.75E+11	2.97E+09	1.08

Category	Tool	t_n	t_{thp}	$s[\%]$
SAT	MiniSat	8.17	7.03	13.99
SAT	MergeSat	7.94	6.90	13.13
ASP	clasp	3.66	3.29	10.18
MaxSAT	open-wbo	1.19	1.09	8.49
MUS	muser2	4.18	3.97	5.16
HWMC	aigbmc	0.89	0.86	4.11
SWMC	cbmc	0.23	0.22	2.76

Benchmarks disponible sur https://github.com/conp-solutions/thp

Astrophysical simulation code

- Fujitsu A64FX
  - Processeur 64 bit
  - Architecture ARM
  - Conçu pour le calcul parallèle massif
  - Cœur du supercomputer Fugaku
- FLASH: Outil de simulation basé sur Fortran

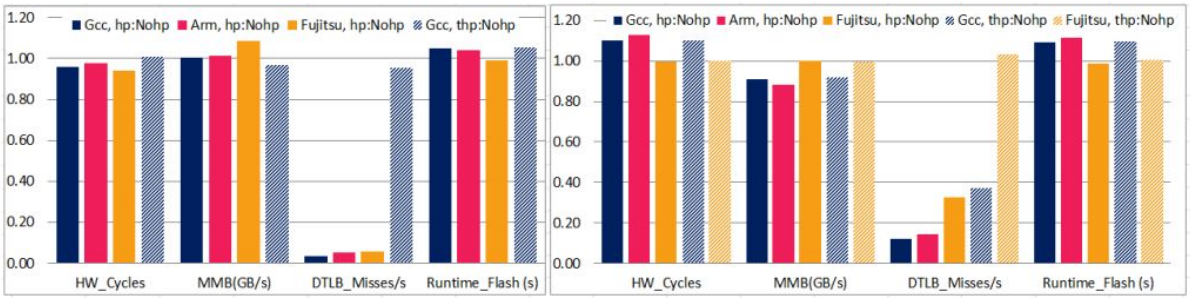
Astrophysical simulation code : Benchmark

- Le but du benchmark est de comparer la performance sur HP et THP et sur différents compilateurs
- Benchmark réalisé sur deux tests astrophysiques :
  - EOS
  - 3-D Hydro
- Outil du testbench : Performance Application Programming Interface (PAPI)

Astrophysical simulation code : Benchmark

Résultat:

1. Influence sur HP/THP est très peu
2. Fujitsu compilateur performe mieux que GCC et ARM compilateur

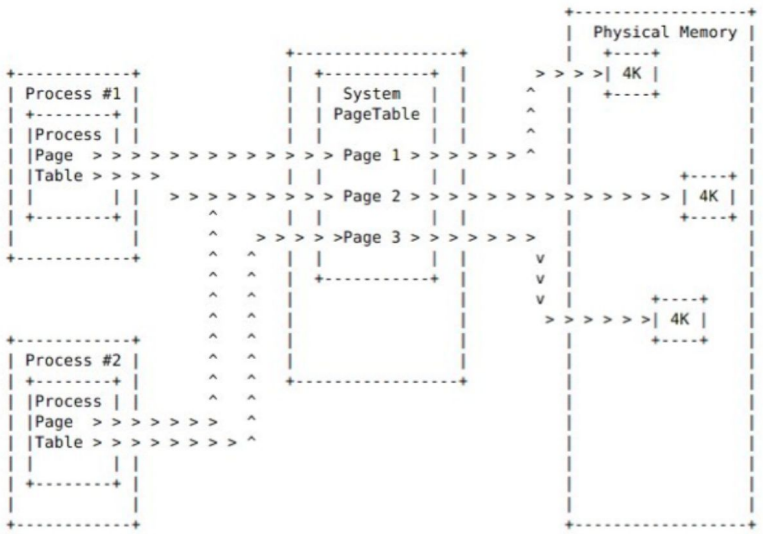


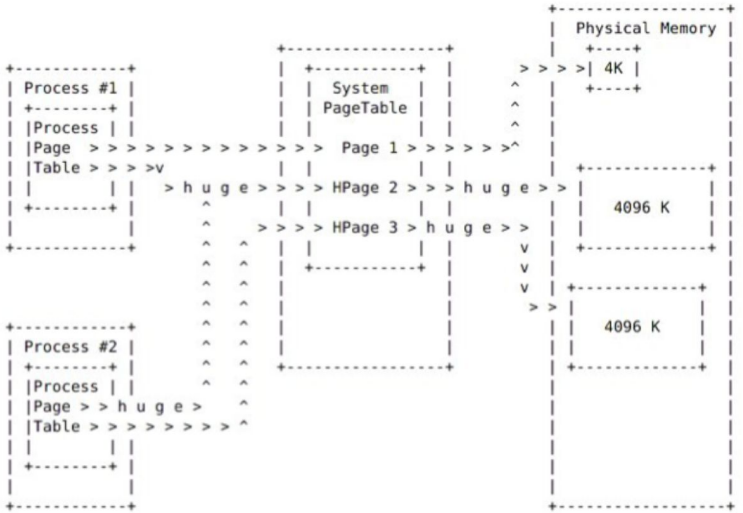
(a) EOS test

(b) 3-d hydro test

Figure 1: Ratios of runs with and without hugepages for each compiler for the (a) EOS test and (b) 3-d hydro test on 1 core

Oracle


Physical Memory
Process #1
Process
Page
Table
System
PageTable
Page 1
Page 2
Page 3
4K
Process #2
Process
Page
Table


Physical Memory
Process #1
Process
Page
Table
System
PageTable
Page 1
HPage 2
HPage 3
huge
4K
4096 K
Process #2
Process
Page
Table

Oracle

- La THP peut entraîner des problèmes de performance dans les scénarios où :
  - La base de données gère des transactions à faible latence
  - Des pages verrouillées avec partage de la mémoire
- La plupart des fournisseurs de bases de données recommandent de désactiver THP (Redis, Oracle)

> *"Because Transparent HugePages are known to cause unexpected node reboots and performance problems with RAC, Oracle strongly advises to disable the use of Transparent HugePages. In addition, Transparent Hugepages may cause problems even in a single-instance database environment with unexpected performance problems or delays. As such, Oracle recommends disabling Transparent HugePages on all Database servers running Oracle."*

Conclusion

Références

- D. P. Bovet and M. Cesati, Understanding the Linux Kernel, 3rd ed. O'Reilly Media, 2005.
- The Linux kernel User’s and administrator’s guide
- Clarifying memory management with page folios, Jonathan Corbet, March 2021
- Chuandong Li, FHPM: Fine-grained Huge Page Management For Virtulization, 2023
- Jie Ren, Dong Xu, Junhee Ryu, Kwangsik Shin, Daewoo Kim and Dong Li, Rethinking Memory Profiling and Migration for Multi-Tiered Large Memory Systems, 2023
- arXiv:2004.14378v1 [cs.LO] 29 Apr 2020
- arXiv:2309.04652v1 [cs.DC] 9 Sep 2023

Questions ?