NUMA systémy se sdílenou pamětí

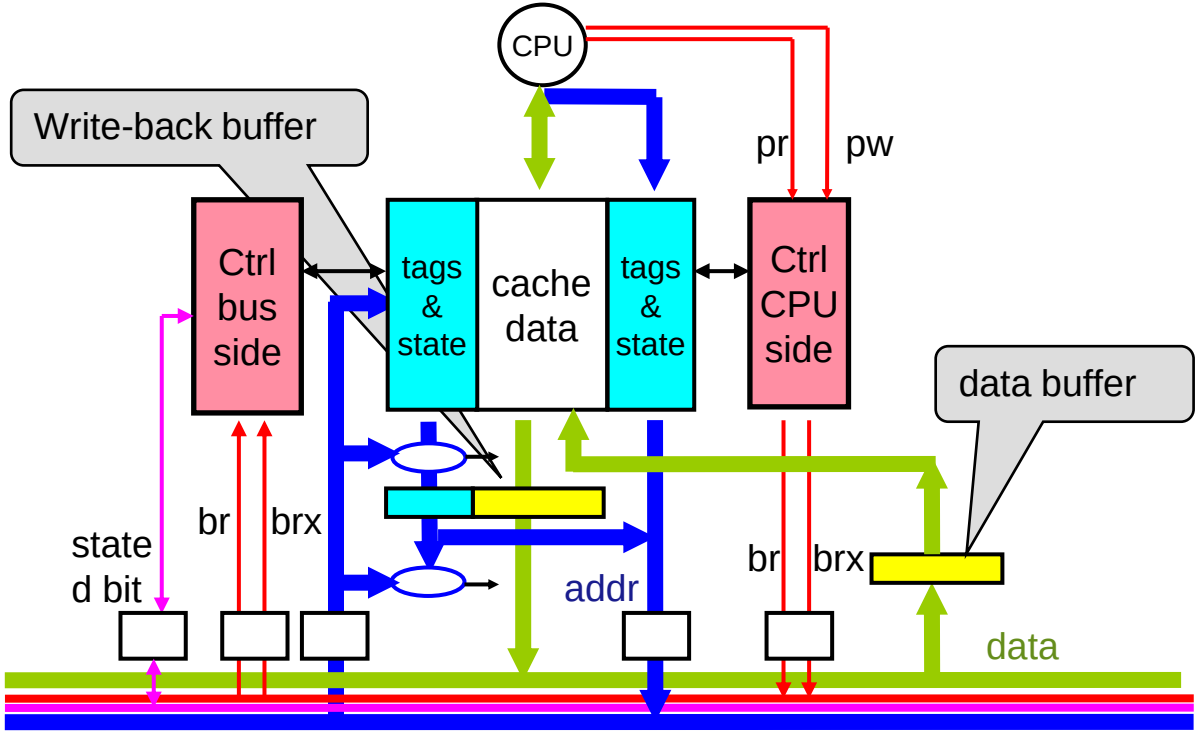
AVS – Architektury výpočetních systémů

Týden 11, 2024/2025

Jirka Jaroš

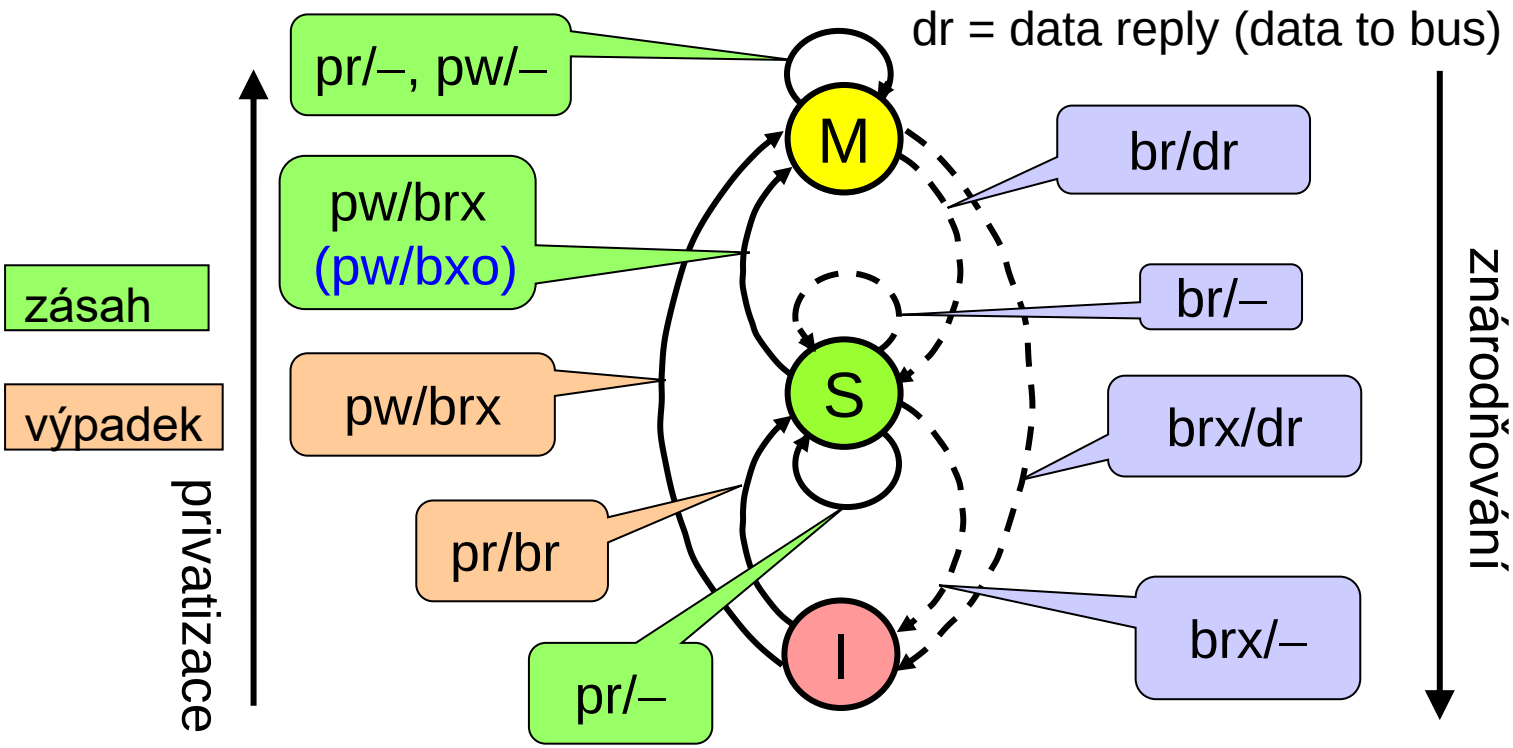
Vysoké učení technické v Brně, Fakulta informačních technologií
Božetěchova 1/2, 612 66 Brno - Královo Pole
jarosjir@fit.vutbr.cz

Řadič (Ctrl) paměti cache s nasloucháním



Celkový stavový diagram protokolu MSI

Tlusté přechody: žadatel, čárkované přechody: ostatní cache



Příklad 1. CC protokol MSI

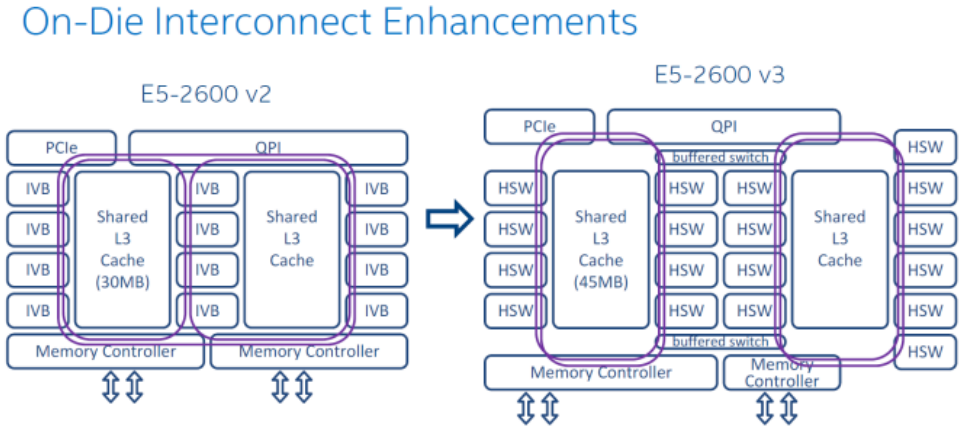
3 procesorový systém se společnou sběrnicí, sdílenou pamětí a koherentními pamětmi cache L1 (protokol MSI) vykonává operace se sdílenou proměnnou u dle tabulky. Vyplňte tabulku pomocí symbolů M, S, I, br, brx, SM, C1, C2, C3, - :

	Akce procesoru	Signály na sběrnici	Stav bloku v C1	Stav bloku v C2	Stav bloku v C3	Data dodá	Data přijme
0.	-	-	M	I	I	-	-
1.	P3 čte u						
2.	P3 píše do u						
3.	P1 píše do u						
4.	P2 čte u						
5.	P3 píše do u						
6a.	P3 čte u						
6b	P3 čte u						

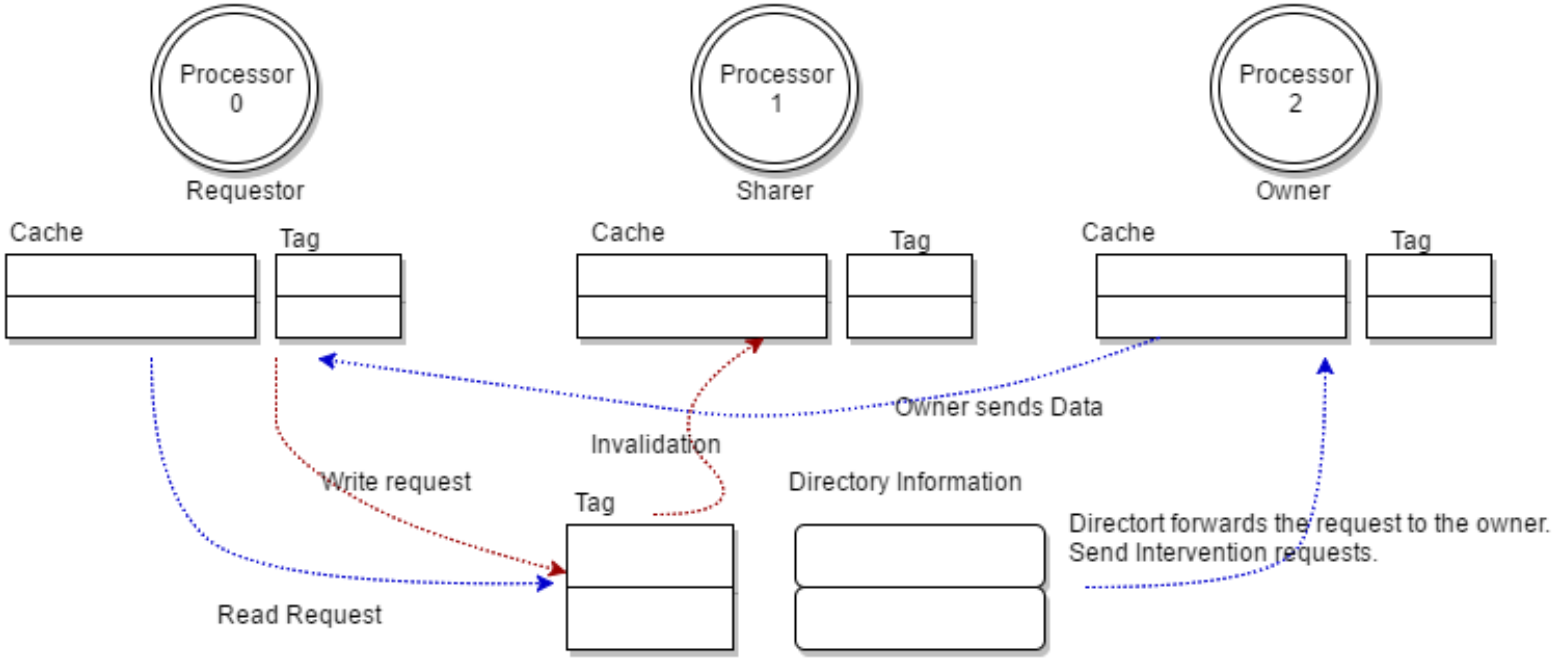
PROTOKOLY CC ZALOŽENÉ NA ADRESÁŘÍCH

CC založená na adresářích

- U CMP se sběrnicí je snadný broadcast signálů *br*, *brx*, *bxo* a komunikace bitů *d*, *e*, *f*. Arbitr sběrnice také žádosti seřazuje.
- U multiprocesorů **NUMA** s větším počtem CMP by byl broadcast složitý. Proto žadatel komunikuje **přes prostředníka** (zástupce). Tímto prostředníkem jsou **distribuované adresáře** (directories) a jejich řadiče (DirCtrl) u jednotlivých modulů DSM.
- **Domovský adresář** (Home Directory, *H*) pro určitý modul SM udržuje informace o blocích (cache line) v modulu (RAM):
  - jestli je blok platný, čistý nebo špinavý,
  - kde se nachází, ve které (**sdílené last level**) cache (LLC).
- Řadič domovského adresáře **DirCtrl:**
  - Seřazuje příchozí žádosti a reaguje na ně.
  - Příkazy (zprávy) pro koherenci, ale místo broadcastu zasílá **pouze do relevantních uzlů** (multicast).
  - **To u velkých systémů enormně redukuje komunikaci.**



Princip CC na systémech s domovskými adresáři



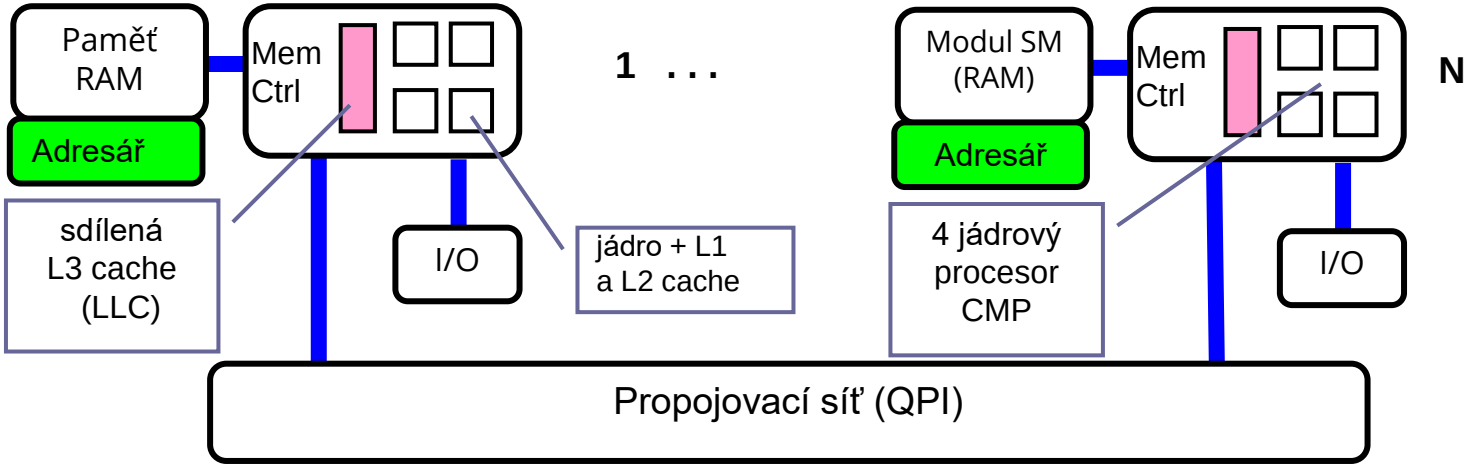
CC založená na distribuovaných adresářích

- **Záznam v adresáři o bloku na adrese *X* je *N*+1 bitový vektor:**
  - nultý clean/dirty bit V_0
  - *N* prezenčních bitů V_i, i =1, 2, …, *N* (bit-mapa):

```
bit    0 1 2 3 4 5 …    N
X  [  0 0 1 1 0 1 … 0 1 ] sdílený blok S je ve více cache
X  [  0 0 0 0 0 0 … 0 0 ] samé nuly: blok je jen v RAM
X  [  1 0 0 1 0 0 … 0 0 ] špinavý blok M je v cache 3
```

- **Bloky v cache** jsou označeny duplicitně dirty/clean bitem (kromě valid bitu).
- Stavové bity jsou čteny a modifikovány dvěma řadiči: **CacheCtrl** a **DirCtrl**. (V názvosloví Intel: „Cache agent“ a „Home agent“ nebo jen „Home“). Zpracovávají zprávy CC a datové odpovědi.

Malý multiprocesor DSM CC-NUMA založený na adresářích

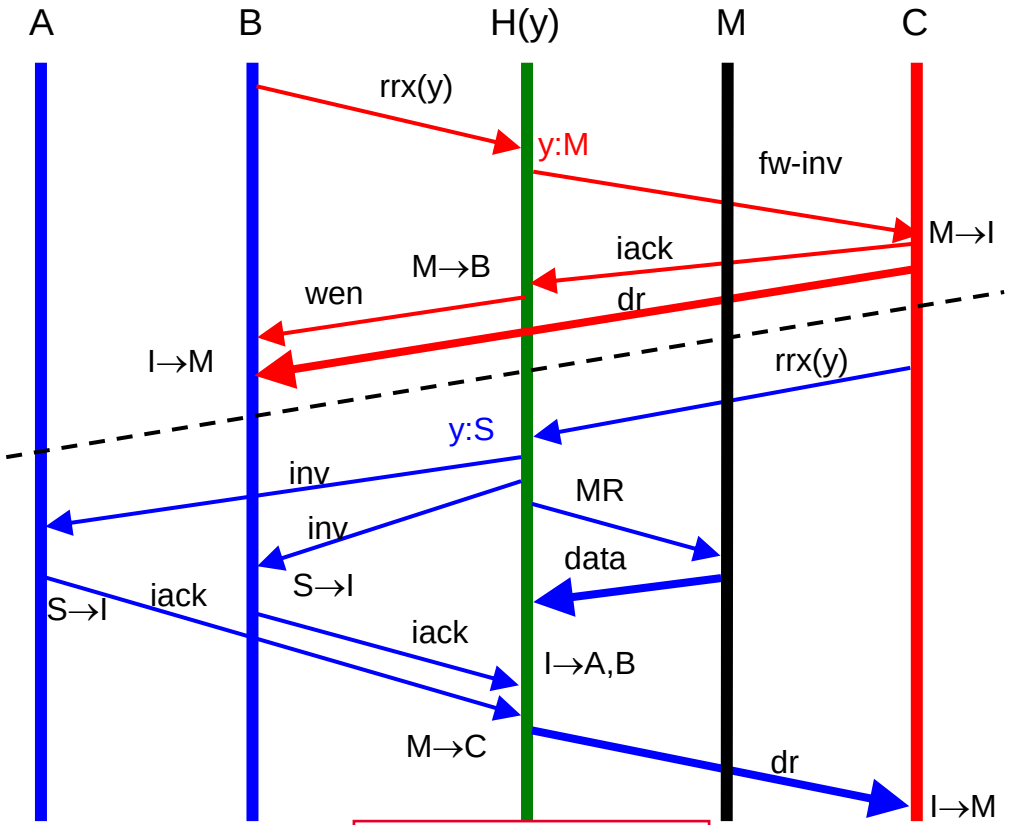


Sekvence komunikací u protokolu CC s adresáři:

1. žádost R/W → Home
2. Home kontaktuje jen relevantní agenty CacheCtrl
3. CacheCtrl: odpověď (potvrzení)→ Home, datová odpověď žadateli, případně i do Home
4. Home: povolení R/W → žadateli

- *Řadiče (agenti) DirCtrl a CacheCtrl modifikují podle potřeby stavové bity bloků.*
- Všechna data procházejí přes cache L1. Mohou být při nedostatku místa vyhozeny do vyšší úrovně cache nebo až do SM RAM.

Ukázka: Výpadek při zápisu do bloku y (M a S)



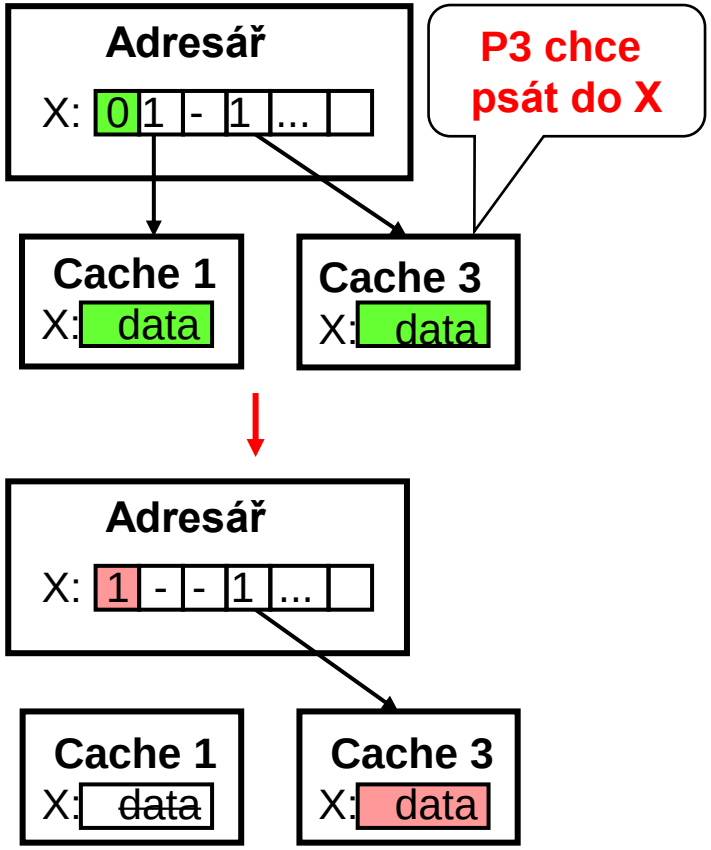
Protokol MSI

Legenda:

rrx	= rr exclusive
fw-inv	= forward-invalidate
dr	= data reply
wen	= write enable
iack	= invalidate acknowledge
inv	= invalidate
fw	= forward and make shared
wb	= write back do paměti

A, B	= cache agenti (řadiče LLC procesorů)
H(x)	= home agent bloku x (DirCtrl)
M	= řadič paměti MemCtrl
MR	= memory read

Implementace distribuovaných adresářů



Adresář s úplnou bitovou mapou: ke každému bloku v paměti je přiřazen

- bit-vektor *N* bitů prezence
- 1 clean/dirty bit.

Režie na 1 blok 64 byte, který má kopie až na *N* procesorech (v LLC) :

- *N* = 8: 9 bitů / (64 * 8) = 2%
- *N* = 64: 65 bitů / (64 * 8) = 13%
- *N* = 256: 257 bitů / (64 * 8) = 50%.

Není škálovatelné! Celková paměťová **režie** = (*N* + 1) bitů * *N* * $M_{local} \approx O(N^2)$, kde M_{local} je počet bloků v 1 modulu DSM. Používá se do *N* = 64.

Omezené a vázané adresáře

Omezené adresáře:

- Sdílení bloku je omezeno na max. $Q < N$ lokací.
- Je-li třeba víc než Q kopií, bude při dosažení max. počtu Q kopii třeba někomu kopii zneplatnit a jinému přidělit.

```
#pragma omp parallel for shared(N, a)
for (int i = 0; i < N; i++) a[i]++
```

Vázané adresáře:

- Adresáře jsou zabudovány do samotných pamětí cache ve formě obousměrného vázaného seznamu kvůli snadnému vkládání a odebírání položek.
- **Režie**: 2 ukazovátka $\log_2 N$ bitů na jeden blok v každé cache.
- **Škálovatelné** (standard SCI = Scalable Coherent Interface).
- **Nevýhoda**: invalidace vyžaduje průchod celým seznamem.

Koherence ve zvláštních případech: TLB

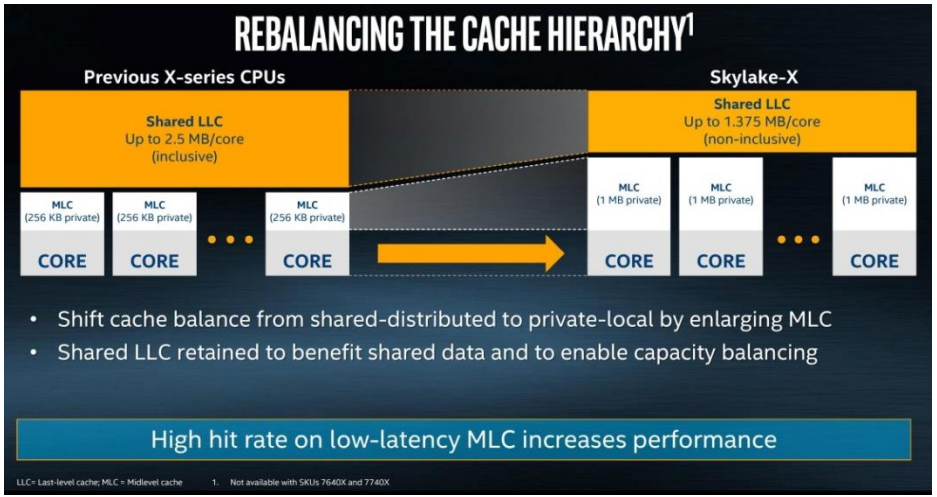
- **OS udržuje tabulku stránek PT ve sdílené paměti. Položky PT jsou pro rychlý překlad VA → PA v malé cache TLB na čipu.**
  - položky vkládá do TLB HW nebo OS.
  - Stránka (z disku) se načte do modulu sdílené paměti toho procesoru, jehož jádro se jí dotkne první (first-touch) tam bude její „Home“.
  - Pokud chce jiné jádro přistupovat do stejné stránky, bude generovat výpadek TLB.
  - Po aktualizaci TLB z PT načte data z modulu SM Home do své cache.
  - Stejná položka se tak může vyskytovat v TLB více jader. Musí se proto řešit koherence všech TLB, např. v případě výměny stránky nebo změny přístupových práv.
- **Řešení:**
  - použití části D-cache pro TLB
  - invalidace položek TLB přes OS pomocí přerušení
  - HW instrukce TLB Invalidate Entry (PowerPC)

Koherence ve více úrovních cache

- V dnešních vícejádrových procesorech bývá sdílená Last Level Cache LLC = L3 (několik MiB až desítek MiB); nově i sdílená embedded eDRAM L3 nebo L4 (96–128 MiB).
- Koherence mezi několika vícejádrovými čipy (CMP) je zajišťována přenosem spec. paketů (součást protokolu QPI). Intel QPI implementuje 5 stavový protokol MESIF.
- Koherenci na **nižších úrovních** (změny stavů bloků, wb, datové odpovědi) vyřídí u inkluzivních **cache LLC** její řadič (Cache agent).
- Pokud jsou nižší cache exkluzivní, musí se prohledávat stav všech nižších cache u všech jader! Bloky se navíc musí vyměňovat.

Inkluzivní cache (preferuje Intel do Skylake-X)

- Je-li čistý blok přítomen v L1, musí být též v L2.
- Je-li v exkluzivním vlastnictví L1 špinavý blok, musí být alokován a zneplatněn i v L2.
- Nejsou-li data v **inkluzivní sdílené L3, nejsou ani v L2 i L1** cache žádného jádra, takže se rovnou generuje požadavek do RAM (nemusí se hledat po jádrech). Najdou-li se data v L3, přečtou se přímo odtud.
- U bloků v L3 je indikace, ve kterých jádrech je blok přítomen
  - (1 inclusion bit na 1 jádro, malá bit-mapa) – takový malý adresář.
  - To dovoluje filtrovat žádosti *invalidace* přicházející do LLC = L3 a nepropustit ty, co se v L1 a L2 nemohou uplatnit.
  - Pak lze vystačit pouze s jedním řadičem koherence pro mateřskou cache (L3) a dceřiné cache (L2, L1).
- Udržování inkluze je za určitých podmínek samočinné



Exkluzivní cache (preferuje AMD)

- Data jsou v L1 nebo v L2, ale nikdy ne v obou
  - sdílená LLC = L3 není ani čistě exkluzivní ani inkluzivní – dnes stejné jako Intel.
- Nejsou-li data v L1 až v L3, musí se prohledávat stav cache ostatních jader. Teprve při neúspěchu se jde do RAM.
- Když je v **L1 výpadek a v L2 zásah**, musí se blok mezi **L1 a L2 vyměnit**, což je více práce než jen kopírovat blok z L2 do L1 u inkluzivní cache.
  - Navíc bloky L1 a L2 musí mít pro výměnu stejnou velikost.
- Systém exkluzivních cache uchovává více dat než inkluzivní. To je znát hlavně když L2 a L3 mají podobnou velikost.

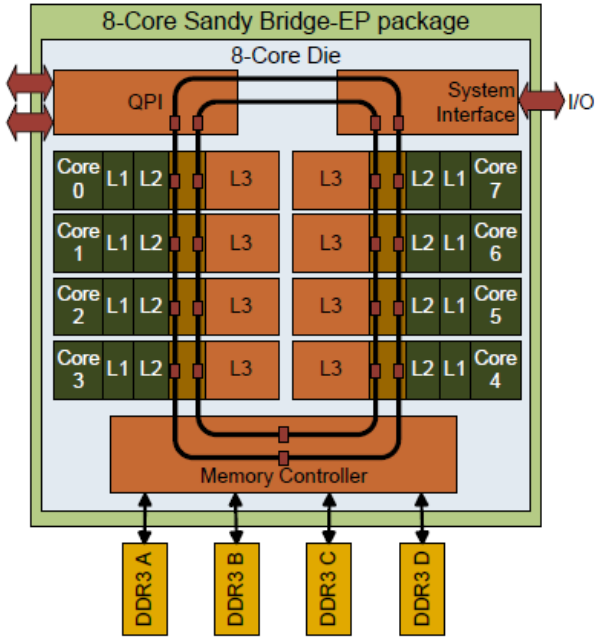
Cache Hierarchy	AMD EPYC 7742 DDR4-3200	AMD EPYC 7601 DDR4-2400	Intel Xeon 8280 DDR-2666
L1 Cache	32KB 4 cycles	32KB 4 cycles	32KB 4 cycles
L2 Cache	512KB 13 cycles	512KB 12 cycles	1024KB 14 cycles
L3 Cache	16MB / CCX (4C) 256MB Total ~34 cycles (avg)	16MB / CCX (4C) 64MB Total	38.5MB / (28C) Shared ~46 cycles (avg)

Alternativní přístupy k CC

- NCC-NUMA – Cache bez HW podpory koherence,;
  - Pouze instrukce a privátní data mohou být v cache.
  - Sdílená data jsou kompilátorem označena jako neschopná modifikace v cache (CRAY T3D, Intel SCC – Single Chip Cloud Computer).
- Bez pamětí cache
  - Latence paměťových přístupů skryta multi-threadingem (Cray MTA, Multi-Threaded Architecture, GPGPU = GPU pro univerzální výpočty).
- **SVM** – Shared Virtual Memory
  - Používá mechanismus virtuální paměti s podporou OS.
  - Místo bloku stránka, OS označuje stránky RW, RO, INV; místo výpadků bloků výpadky stránek. Cílem není HPC.
- **Transakční paměť** (Transactional Memory)
  - Programátor specifikuje začátek a konec transakce ve zdrojovém kódu a TM systém provádí transakce paralelně a spekulativně – optimisticky předpokládá, že budou provedeny atomicky.
- **Pokud se z SM pouze čte nebo do ní pouze zapisuje, není třeba koherence**
  - Mám tedy zdrojové a cílové pole, a před přehozením pointerů zavolám FLUSH

PROTOKOLY CC SOUČASNÝCH PROCESORŮ

Příklad koherence na kružnici: Intel Sandy Bridge



- Jádra, grafika, sdílená L3 cache (20 MiB), 4 kanálový MemCtrl a 2 linky QPI jsou propojeny 4 kružnicovým propojením s propustností 256 bit/takt (šířka AVX registru).
- L3 cache je inkluzivní pro cache L1 a L2 všech jader, rozdělená do řezů. Fyzické adresy rozděleny na řezy jednou hash funkcí. Každý řez má CacheCtrl a inkluzivní bity.
- Zprávy CC mezi čipy: QPI na 8 GT/s.

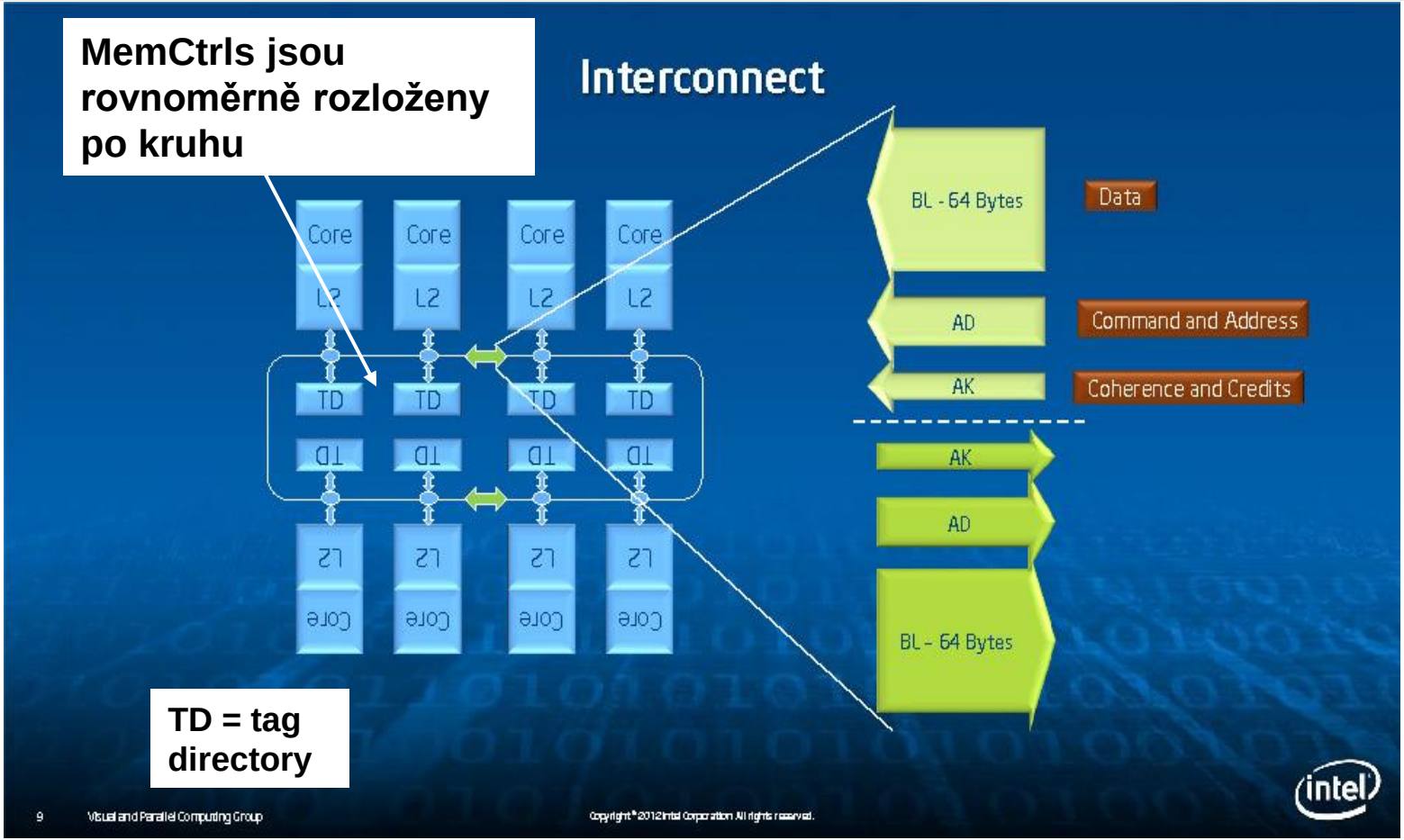
Přístup na prstenec podléhá arbitráži. Vysílání do prstence může být překryto s příjmem.

Pokud vysílá víc než 1 uzel, zprávy nejsou totálně uspořádané.

CC protokol se „seřazovacím bodem“

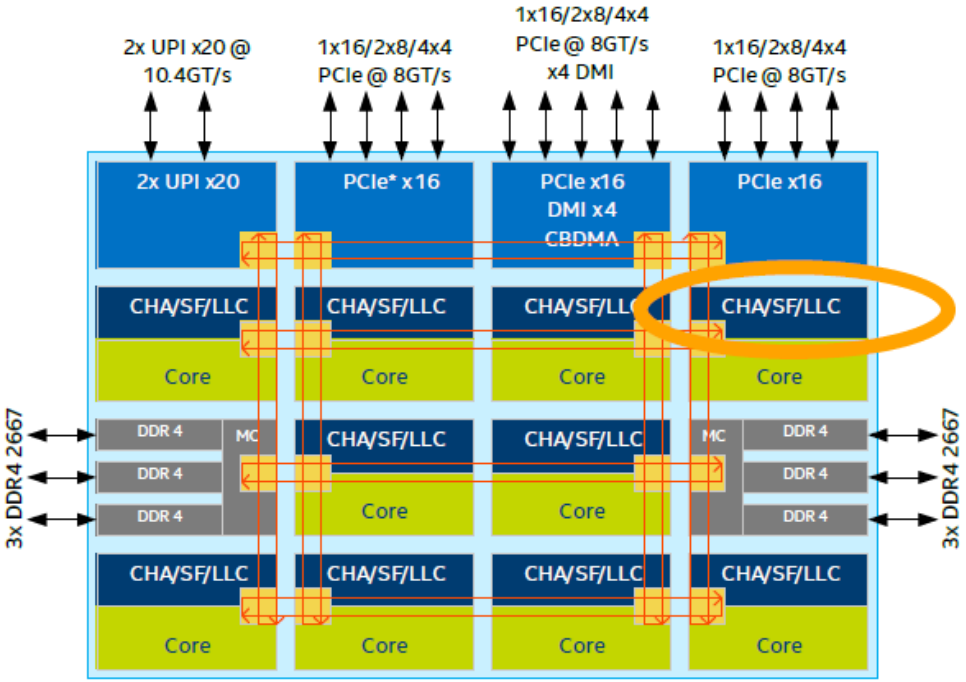
- Žádost **rr** nebo **rrx** putuje nejdříve od žadatele do seřazovacího bodu (MemCtrl) a tam je teprve aktivována (1 bitem v hlavičce).
- Pak proběhne celá rotace (sbírání odezev uzlů – data od majitele, iack)
- Seřazovací bod pak vyjme zprávu a pošle povolení zápisu (wen) žadateli. Průměrně P/2 + P + P/2 = 2P hopů.

- **Interface na DRAM:**
  MemCtrl na žádost rr nebo rrx načte a dodá blok dat z paměti jen je-li to nutné, jinak blok ve stavu M, E, O (F) dodá rychleji přímo vlastník.
- MemCtrl může též spekulativně načítat blok vždy a po vyhodnocení odezev jej předat žadateli na jeho explicitní žádost.


MemCtrls jsou rovnoměrně rozloženy po kruhu
Interconnect
Core
L2
TD
BL - 64 Bytes
Data
AD
Command and Address
AK
Coherence and Credits
AK
AD
BL – 64 Bytes
TD = tag directory
9
Visual and Parallel Computing Group
Copyright © 2012 Intel Corporation. All rights reserved.
intel

Distribuované cache a home agent na Skylake-X

- Eliminates large tracker structures at memory controllers, allowing more requests in flight and processes them concurrently
- Reduces traffic on mesh by eliminating home agent to LLC interaction
- Reduces latency by launching snoops earlier and obviates need for different snoop modes.



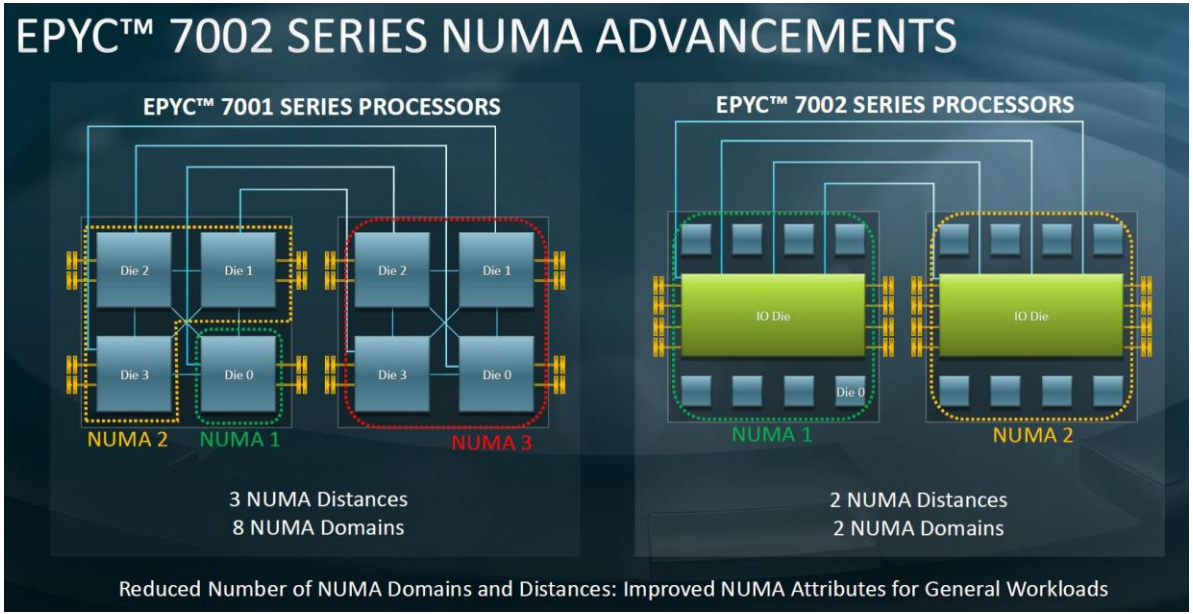
ARCHITEKTURY S DISTRIBUOVANOU SDÍLENOU PAMĚTÍ (NUMA DSM)

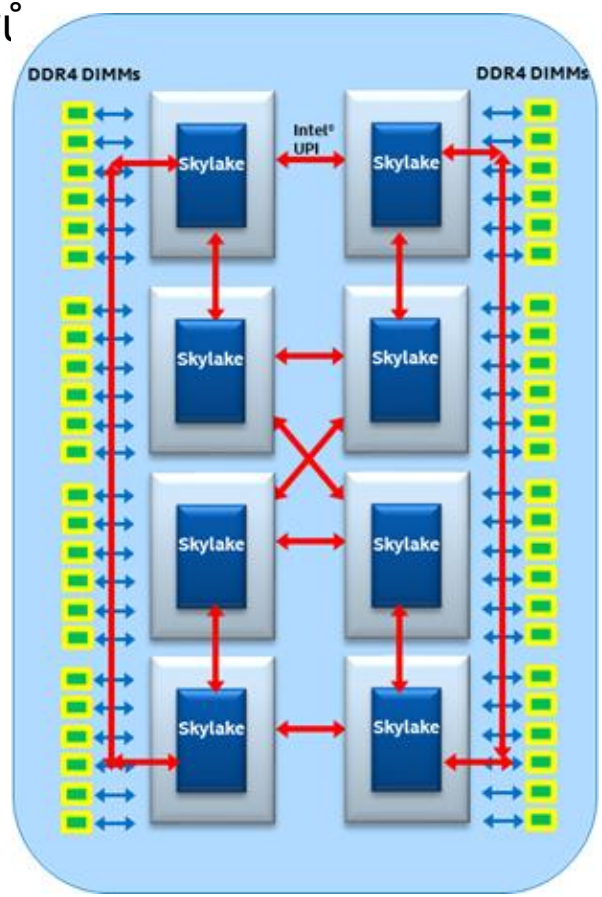
Architektura DSM – Distributed Shared Memory

- Paměti cache s plnou hardwarovou podporou koherence.
  Jeden protokol CC zadrátován do systému.
- Paměť fyzicky distribuovaná na uzly, fyzický adresový prostor je logicky sdílený – sítí putuje žádost s fyzickou adresou PA.
- Propojovací síť místo sběrnice obecně znamená NUMA (NonUniform local and remote Memory Access) ⇒ rozhlašování (multicast) je obtížnější (kdy je hotovo?)
- Paměťové transakce rozdělené, na každou žádost je nutná odpověď / potvrzení (ACK)
- Citlivost na distribuci dat, optimálně data alokovat v uzlu, kde se s nimi pracuje. (U centralizované SM je alokace dat libovolná.)
- Škálovatelnost do stovek až 2 tisíců jader.

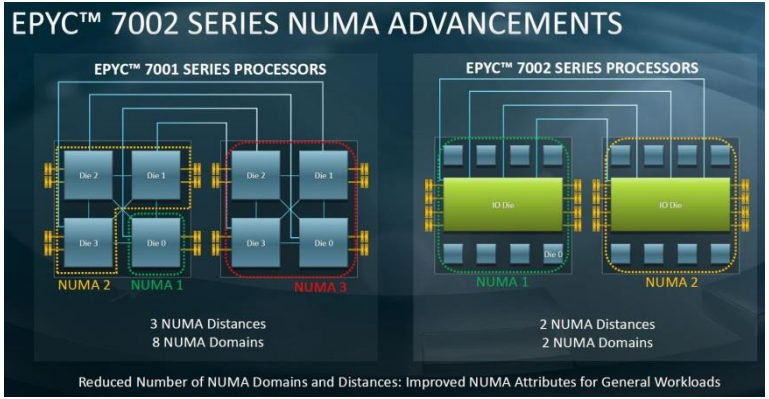
NUMA systémy na jedné základní desce

- Intel UPI má 2 nebo 3 linky pro propojení procesorů
- AMD Infinity fabric pro propojení procesorů.
- AMD snižuje počet NUMA regionů vyjmutím řadiče paměti z procesoru, zvyšuje latence.





Latence a propustnosti pamětí a cache

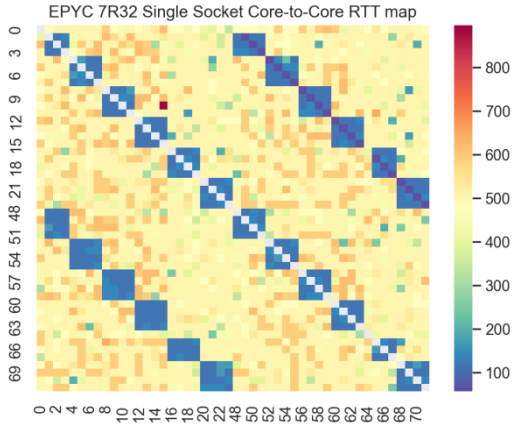


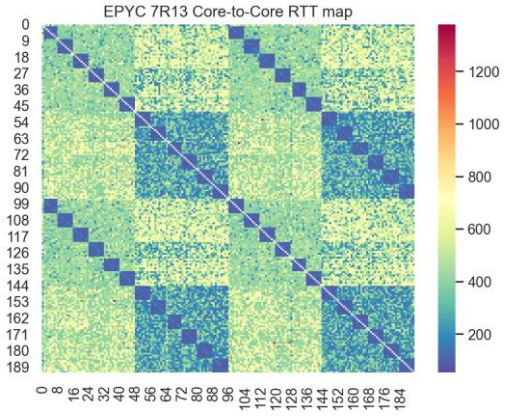
AMD EPYC Infinity Fabric Memory Bandwidth DDR4 2666 (in MB/s)								
	0	1	2	3	4	5	6	7
0	36346	19101	19910	19913	8143	8080	***8143***	8159
1	19103	36435	19913	19911	8080	8080	8084	***8066***
2	19905	19916	36280	19065	***8070***	8049	8088	8050
3	20095	19908	19093	36357	8055	***8045***	8055	8055
4	8072	8069	***8062***	8084	36409	19126	19931	19932
5	8083	8060	8086	***7996***	19125	36389	19927	19935
6	***8170***	8124	8110	8194	19933	19929	36479	19120
7	8123	***8094***	8169	8088	19938	19941	19101	36409

© 2017 ServeTheHome.com

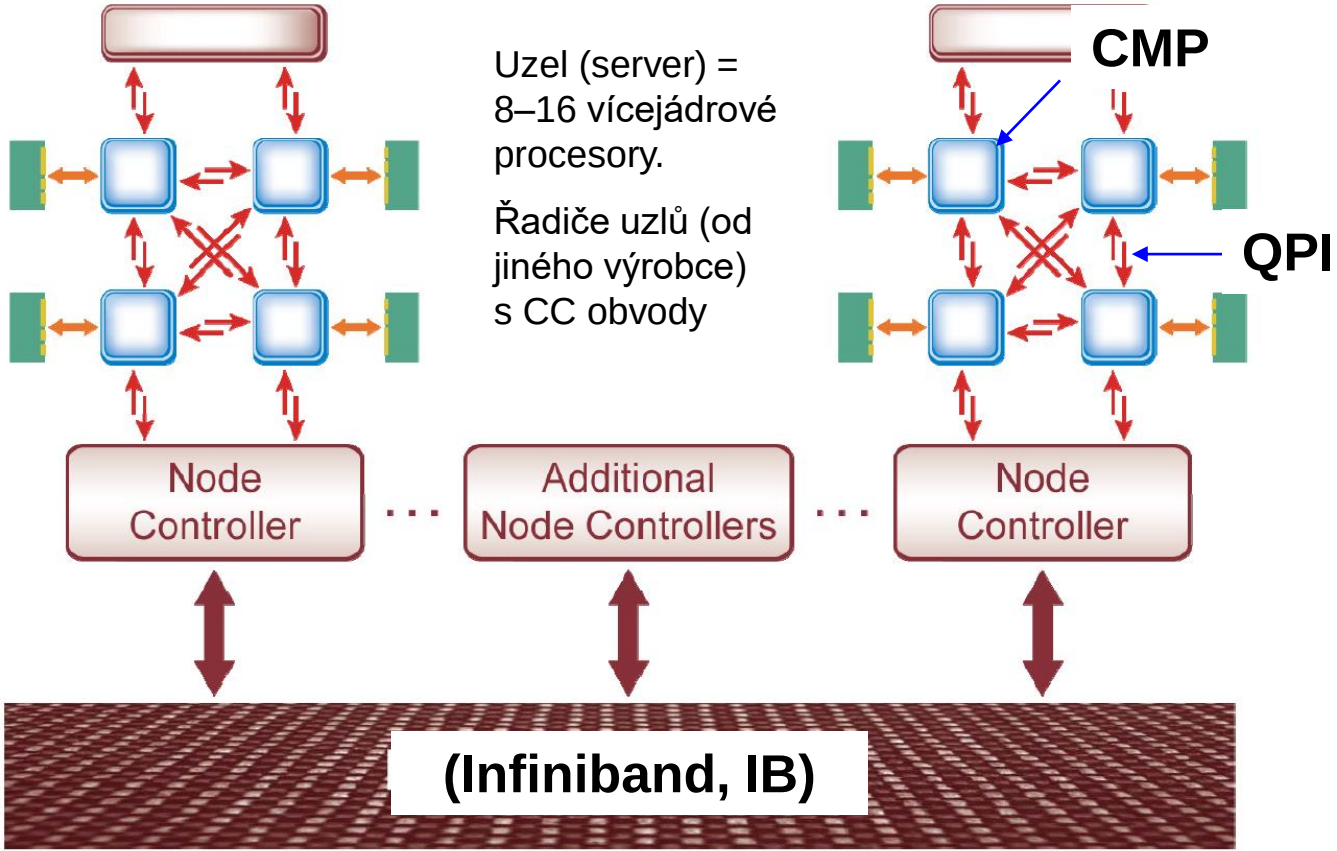
AMD EPYC Infinity Fabric DDR4 2666 Idle Latencies (in ns) NUMA Node Interconnects (in ns)								
	0	1	2	3	4	5	6	7
0	81	138	133	133	242	241	***200***	235
1	137	85	132	133	241	241	235	***199***
2	133	132	84	137	***199***	246	234	234
3	133	133	137	84	245	***198***	234	234
4	241	241	***199***	235	85	137	133	133
5	241	241	237	***198***	137	85	133	133
6	***199***	246	234	234	133	133	85	137
7	245	***199***	234	234	133	133	137	84

© 2017 ServeTheHome.com





Větší systémy Intel CC-NUMA propojené sítí



Řešení koherence v CC-NUMA

- Domovský adresář řadí příchozí žádosti. Nemusí vědět, kdy dokončí všechny transakce ve všech uzlech.
- Ve frontě žádostí v domovském adresáři jsou použity přídavné stavy „busy“ nebo „rozpracovaný“ (pending)
- Při indikaci že operace je v běhu, musí zpozdit další operace na stejné adrese; může se zatím zpracovat transakce pro jiný blok. Provedení:
  - bufer rozpracovaných žádostí v domovském uzlu
  - bufer žádostí u žadatele
  - (bez buferování žádostí): NACK a zkusit znova
- CC protokoly:
  - s distribuovanými adresáři
  - s naslouchánim – oddělené sítě pro adresování (podpora broadcastu) a data (např. X-bar, kružnice)

coherence state	latency in ns							
	local		on-chip			other socket		
	L1	L2	L1	L2	L3	L1	L2	L3
modified	1.5	4.6	40.4	38.1	15	123	140	
exclusive			33.8			87.3		
forward			15					
shared								

Latencies for accesses to various memory locations on the dual socket Intel Sandy Bridge system

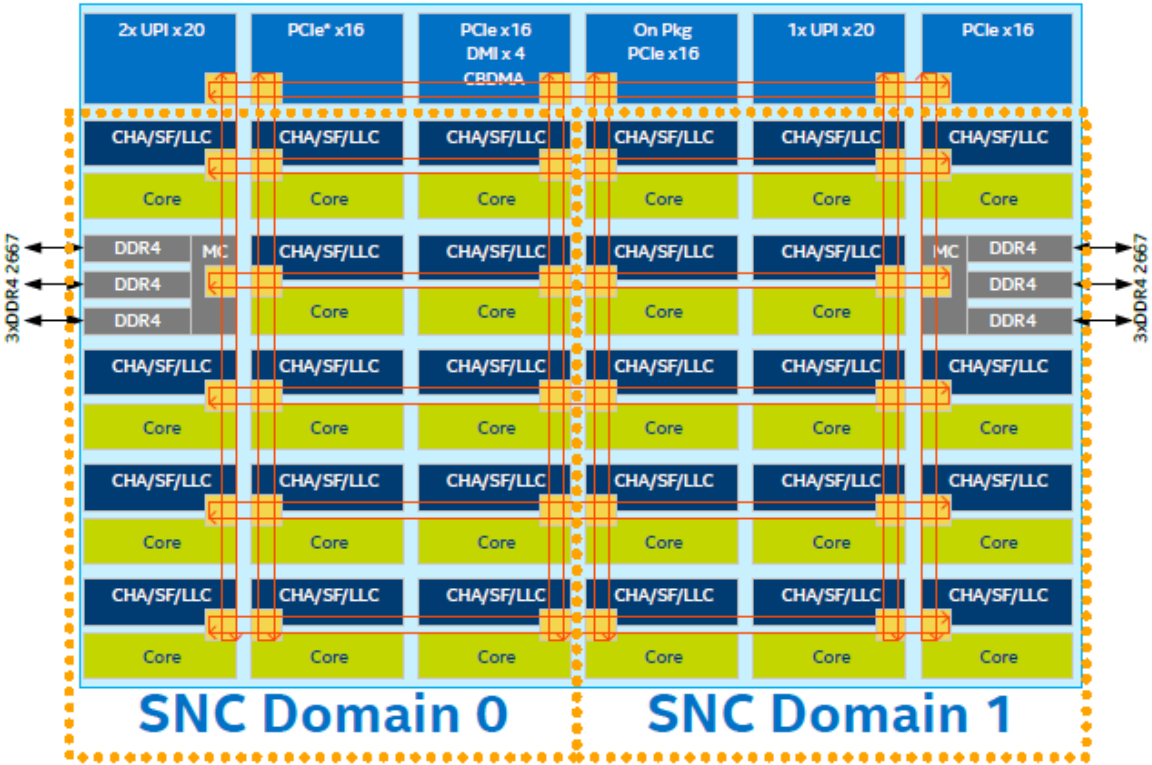
DRAM
local: 81 ns
remote: 133 ns

mem-bind	hops	node1 idle		node1 active	
		1 thrd	8 thrd	1 thrd	8 thrd
node0	0	11.7	39,7	12.9	42.2
node1	1	7.7	18.7	8.6	23.6

Read bandwidths in GB/s for one or eight cores running on node0 reading memory from different nodes on the two socket Intel Sandy Bridge system

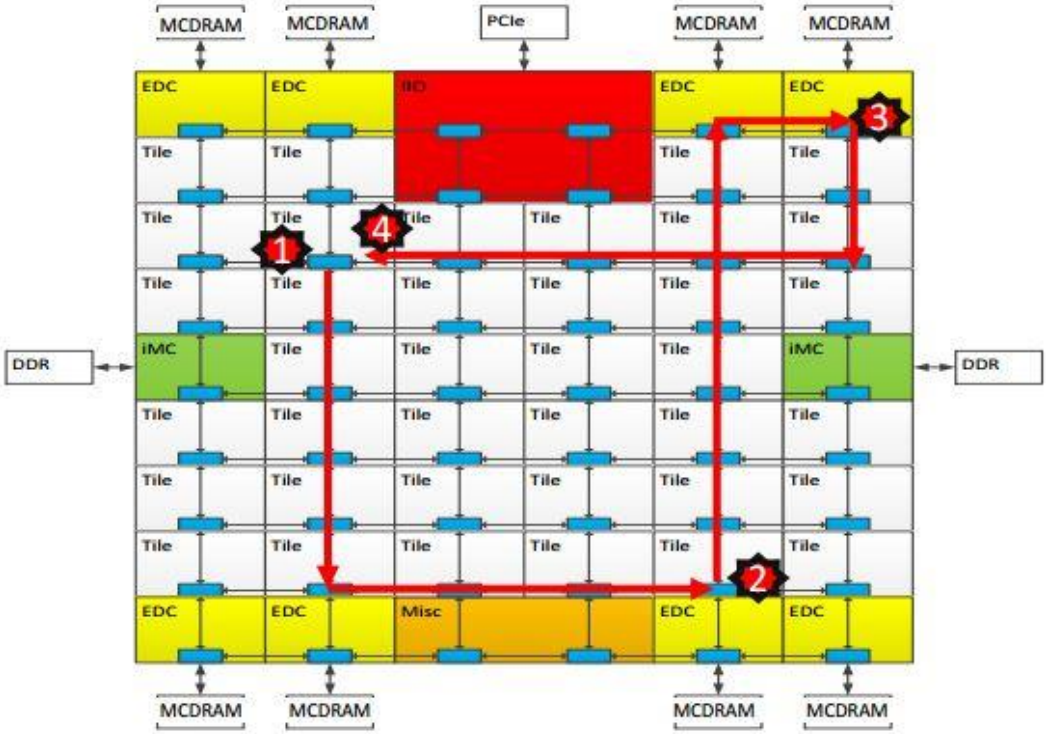
DRAM
local: 11,7 GB/s
remote: 7,8 GB/s

Sub-NUMA Cluster (SNC)



- Sunny Cove umožnuje rozdělit procesor na dvě NUMA domény
  - V obou případech stačí jeden UPI agent pro komunikaci s dalšími procesory
  - Latence pro přístup do paměti je nižší, pokud programátor ví co dělá.
  - Kapacita LLC je využita lépe (méně kopií téhož bloku).

Cluster Mode: All-to-All



Address uniformly hashed across all distributed directories

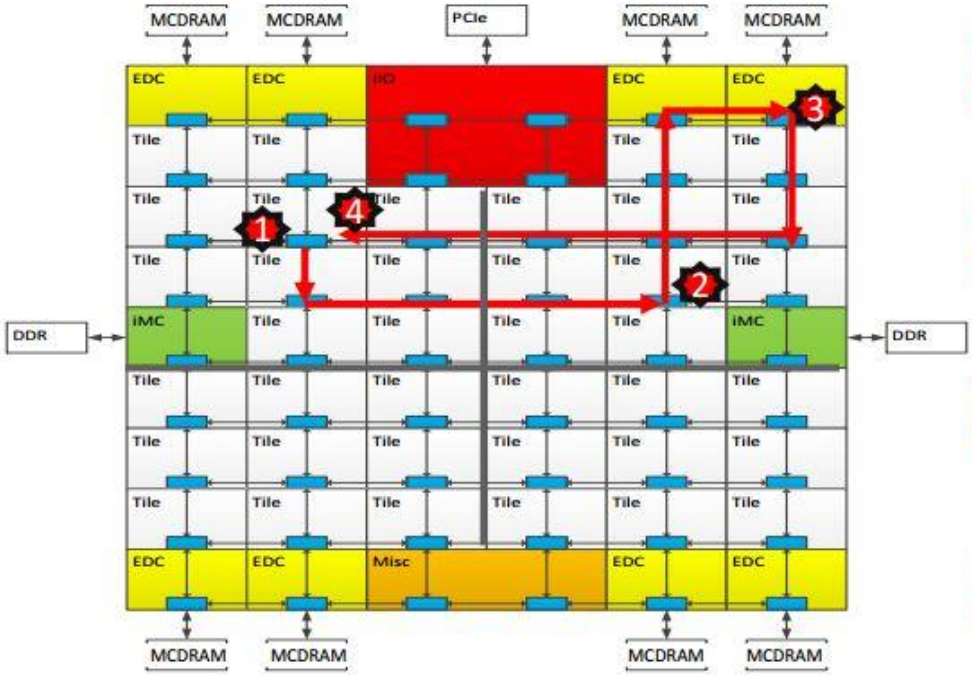
No affinity between Tile, Directory and Memory

Most general mode. Lower performance than other modes.

Typical Read L2 miss

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

Cluster Mode: Quadrant



Chip divided into four virtual Quadrants

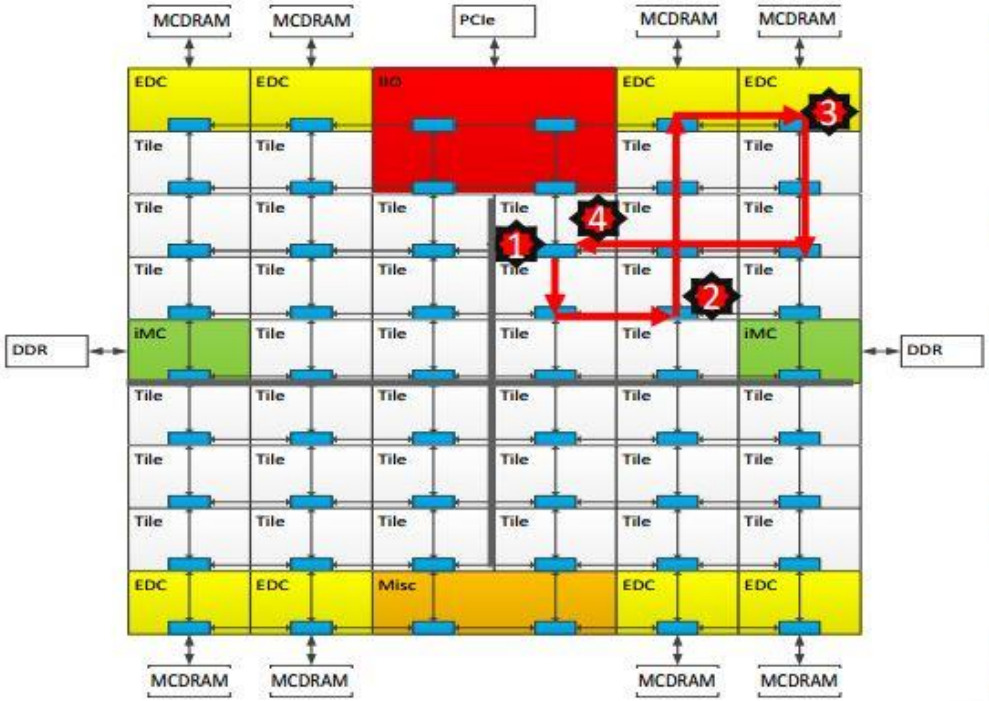
Address hashed to a Directory in the same quadrant as the Memory

Affinity between the Directory and Memory

Lower latency and higher BW than all-to-all. SW Transparent.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

Cluster Mode: Sub-NUMA Clustering (SNC)



1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

Each Quadrant (Cluster) exposed as a separate NUMA domain to OS.

Looks analogous to 4-Socket Xeon

Affinity between Tile, Directory and Memory

Local communication. Lowest latency of all modes.

SW needs to NUMA optimize to get benefit.

Jak zjistit hardwarovou topologii a kontrolovat NUMA

- Jak zjistit topologii
  - `$ hwloc-ls --output-format txt`
- Bindování procesů na NUMA domény: numactl
  - *`# only use cores from node 0`*
  - *`# allocate all memory on node 1`*
  - `$ numactl --cpunodebind 0 --membind 1 ./my-program`
- Programově: libnuma

```
int* memory = ...;
int node = 1;
move_pages(0, 1, &memory,
           &node, &status, 0);
```

- Pomocí OpenMP proměnných prostředí a NUMA funkcí

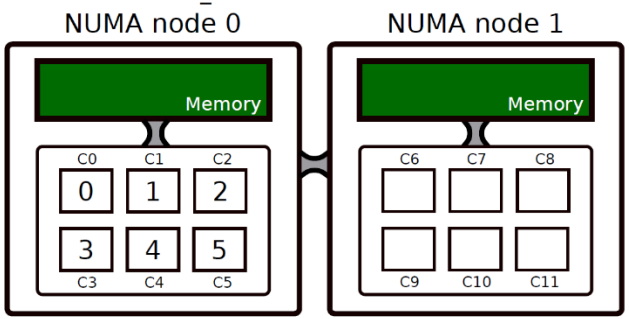
Jak kontrolovat umístění procesů pomocí OpenMP

- **`$OMP_PROC_BIND`** - binds threads to a specific *place*
  - false — do not bind
  - true — pin threads to a single core
  - master — place threads at the same *place* as master thread
  - close — place threads close to master
  - spread — spread threads evenly

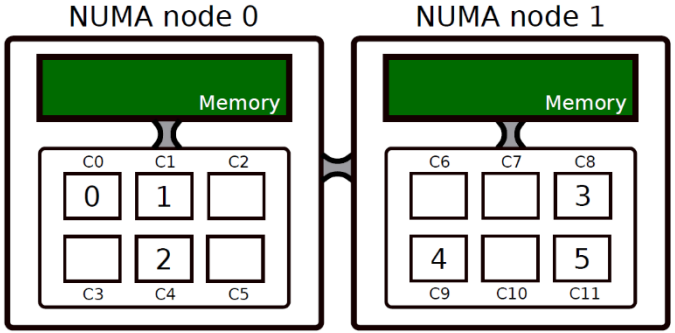
- **`$OMP_PLACES`** - defines what is a *place*
  - threads — *place* is a hardware thread
  - cores — *place* is a core
  - sockets — *place* is a socket (~NUMA node)

Ukázka OpenMP bindování

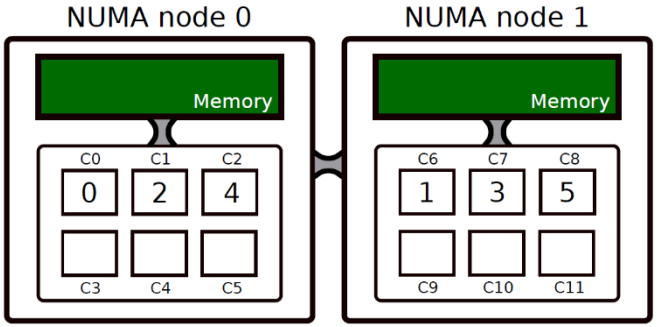
- OMP_NUM_THREADS=6
- OMP_PROC_BIND=close
- OMP_PLACES=cores



OMP_NUM_THREADS=6
OMP_PROC_BIND=close
OMP_PLACES={0,1,4,8,9,11}



- OMP_NUM_THREADS=6
- OMP_PROC_BIND=spread
- OMP_PLACES=cores



Mapování dat v NUMA architektuře

- **NUMA FIRST TOUCH – První přístup do stránky rozhodne o umístění do paměti, která je nejblíže jádru které způsobilo výpadek**

```
const int SIZE = 1024 * 1024;
float* array = malloc(SIZE * sizeof(int));

#pragma omp parallel for schedule(static)
for (auto i = 0; i < SIZE; i++)
  array[i] = 0.0f;


#pragma omp parallel for schedule(static)
for (auto i = 0; i < SIZE; i++)
  processElement(array, i);

free(array);
```

Alokace

First touch mapování do RAM

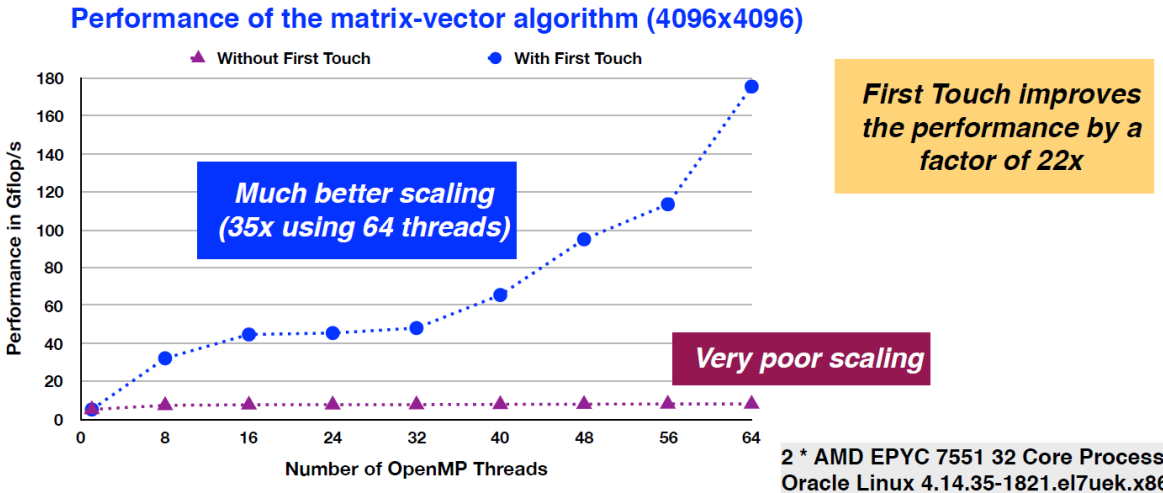
Použití

Problémy s First Touch

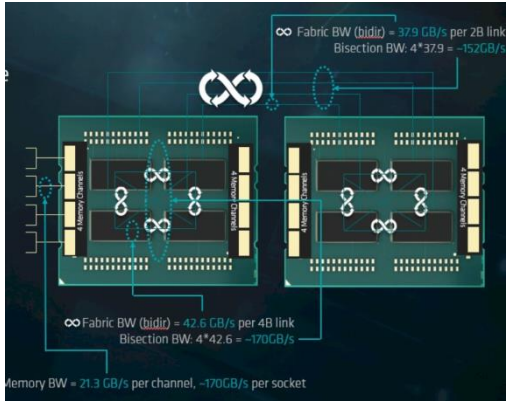
- **Paměť na NUMA doméně 0 bývá obsazena IO buffery**
  - Uživatelská data mohou přetéct do jiné NUMA domény
  - Diskové a síťové operace by měly jít z domény 0
  - Připojená GPU by měla být obsluhována z jader, která mají přímý přístup k danému PCI

- **Přístup k datům musí vykazovat stejný vzor jako při First touch (statické plánování)**
  - Distribuce dat probíhá na úrovni stránek (4 KB, ale 1 GB).
  - Dynamické plánování může poškodit afinitu vláken k datům.
  - Tasky mohou poškodit datovou lokalitu (untied).
  - Různé přístupové vzory v různých částech algoritmu (po řádcích, po sloupcích, atd.).

Ukázka – AMD EPYC – 2x32 jader, 8x NUMA



First Touch improves the performance by a factor of 22x

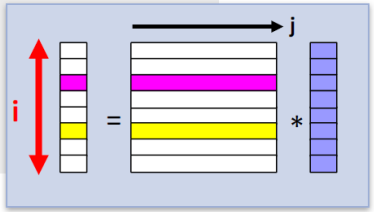


2 * AMD EPYC 7551 32 Core Processor
Oracle Linux 4.14.35-1821.el7uek.x86_64

```
#pragma omp parallel for default(none) \
        shared(m,n,a,b,c)
for (int i=0; i<m; i++)
{
  double sum = 0.0;
  for (int j=0; j<n; j++)
     sum += b[i][j]*c[j];
  a[i] = sum;
}
```

```
$ OMP_PLACES={0}:2:1,{8}:2:1,{16}:2:1,{24}:2:1
$ OMP_PLACES+=,{32}:2:1,{40}:2:1,{48}:2:1,{56}:2:1
$ export $OMP_PLACES

$ export OMP_PROC_BIND=close
```



Pokračování příště