# Empirical Analysis of Coherence for MPSoCs in Avionics Embedded Critical Systems

Marcel Sarraseca Julian
Universitat Autònoma de Barcelona
Email: 1531206@uab.cat

***Resum*—The adoption of complex MPSoCs in avionics embedded critical systems mandates a detailed analysis of their architecture and behavior to facilitate certification.**

**This analysis is hindered by insufficient documentation and the unobvious behavior of some key hardware features.**

**Specifically, the target of this work is the cache coherence protocol of MPSoC T2080 NXP because this is one of the best ways to accelerate the data exchanges.**

**The analysis of the cache coherence protocol consists in making hypotheses with expected behavior. Then with the results of the empirical experiments, we can accept, deny, or modify the initial hypothesis.**

***Paraules clau*—** Multi-Processors System on Chip, MPSoC, NXP, T2080, Cache Coherence Protocol, PMCs, Assembler, Write-Through, Store Gather Buffer, L2 coherence level, avionics, DAL level assurance, critical embedded systems.

## I. Introducció

Actualment, en la indústria de l'aviònica s'està treballant per permetre l'adopció generalitzada de processadors multicore per aplicacions d'aviònica crítiques de nivell A o Design Assurance Level A (DAL A) [1], el qual és el nivell més alt de criticitat (ja que va del DAL E, que és al més baix, al DAL A, que es el més alt). Aquest certificat de seguretat de nivell A de l'estàndard DO-254/ED-80 [2], estipula que certs errors en el hardware de l'electrònica de vol podrien implicar una situació catastròfica, perillosa o greu que provocaria la defunció de totes les persones a bord de l'aeronau [3]. Per aquest motiu la taxa d'errada permesa és molt baixa com es mostra a la Taula I.

| **Design Assurance Level (DAL)** | **Descripció** | **Taxa d'error** | **Sistema d'exemple** |
|---|---|---|---|
| Level A (Catastròfic) | Una falla causa un accident i morts | $< 1 \times 10^{-9}$ probabilitat de falla | Control de vol |
| Level B (Perillós) | Una falla podria causar un accident i morts | $< 1 \times 10^{-7}$ probabilitat de falla | Sistema de frens |
| Level C (Major) | Una falla podria causar estrés i ferits | $< 1 \times 10^{-5}$ probabilitat de falla | Sistema de recolzament |
| Level D (Menor) | Una falla podria causar inconveniència | Sense mètrica de seguretat | Sistema de navegació terrestre |
| Level E (Sense efecte) | No té efectes de seguretat en els passatgers ni en la tripulació | Sense mètrica de seguretat | Entreteniment dels passatgers |

Taula I: Nivells de seguretat DAL [4]

Per una banda, l'adopció de processadors multicore respon a la necessitat dels sistemes d'aviònica de requerir de més capacitat de còmput per a poder executar programes més complexes basats en intel·ligència artificial per a sistemes com flight management system o single-pilot operations[5].

Per altra banda, l'ús de sistemes multicore presenta algunes dificultats pel fet que la seva arquitectura és més complexa que la dels processadors single core i requereix una comprensió profunda del sistema en especial a l'ús de recursos compartits com són els diferents nivells cau i el seu sistema de coherència.

La càrrega de treball de múltiples aplicacions és el millor enfocament per explotar els sistemes multicore, on cada aplicació s'executa en paral·lel per incrementar l'ús dels recursos HW. En l'àmbit d'aplicació el rendiment disminueix respecte a l'execució de cada aplicació en aïllament (en un processador), però en l'àmbit general el rendiment augmenta quan aquestes aplicacions s'executen en paral·lel, això permet reduir els costos de hardware, pes i energia, ja que es pot consolidar més programari en menys maquinari.

Malgrat això, la necessitat d'augmentar el rendiment en l'àmbit d'aplicació és a nivell de fil o thread-level parallelism. L'aplicació és divideix en diverses parts i cada part s'executa en un thread o fil, els fils s'executen de forma concurrent compartint dades i això permet reduir el temps d'execució de l'aplicació. Un exemple il·lustratiu es que el paral·lelisme a nivell de fil s'ha emprat en la planificació de rutes en 3D i la navegació estèreo a través d'altres funcionalitats en temps real crítiques per a la seguretat [6].

En aquest treball es focalitzarà en la coherència cau com a element bàsic per accelerar l'intercanvi de dades.

La raó de ser del sistema de coherència presenta la solució al següent problema: Donats dos processadors P1 i P2, cada un d'ells té una memòria privada L1 i tenen una memòria compartida.

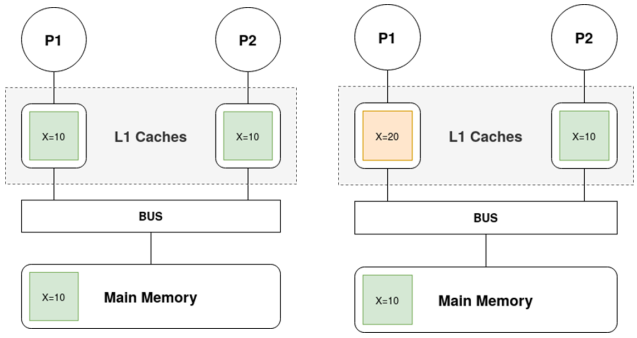
Suposem que $x = 10$ i es troba a la memòria compartida, i els dos processadors fan una lectura i guarden una còpia de $x$ a la seva L1 privada. Els dos cores tindran el mateix valor de $x$, el problema rau en el moment que un dels cores escriu a $x$.

El P1 escriu $x = 20$ en la seva memòria privada L1. El problema és que no hi ha cap forma de saber si el valor de $x$ ha canviat amb lo qual, el P1 treballarà amb el nou valor que té a L1 i P2 treballarà amb la còpia que té a la seva L1 amb el valor $x = 10$ i a la memòria compartida hi continua vigent la còpia $x = 10$, el qual és un valor obsolet i pot generar resultats incorrectes o anòmals en el codi que està executant.

El sistema de coherència cau, permet actualitzar el nou valor de $x$ a la memòria compartida i invalidar la còpia que hi ha a l'L1 del P2 $x = 10$ per posteriorment enviar la còpia vàlida. D'aquesta forma s'assegura que els dos processadors tindran el mateix valor de $x$ [7].

Els sistemes multi-core són més propensos a tenir un rendiment del programa difícil de predir a causa de la seva arquitectura de recursos compartits, ja que el mecanisme de compartició augmenta els retards de temps, en el pitjor dels casos pot provocar un augment del temps d'execució de l'aplicació allotjada, el qual pot tenir conseqüències en sistemes d'aviació [1].

Donat que es requereix un coneixement profund del comportament d'aquest tipus d'arquitectura i que aquestes presenten documentació deficient i poc acurada entre el comportament teòric i el comportament empíric, com és el cas de la placa T2080 de NXP [5], es procedirà a estudiar el seu comportament mitjançant l'experimentació, sota la tutela del departament de Computer Architecture Operative




(a) P1 i P2 llegeixen $x$ (b) P1 escriu a la seva $x$ d'L1

Figura 1: Exemple d'incoherència cau

Systems (CAOS) de la institució Barcelona Supercomputing Center (BSC).

## II. Objectius

L'objectiu d'aquest treball consisteix en la investigació i mesura del comportament del sistema de coherència cau de la plataforma multicore T2080 de NXP a través de l'experimentació i extracció d'evidències empíriques mitjançant els PMCs (Performance Monitor Counters)[8].

Els monitors d'esdeveniments hardware són el bloc principal per proveir d'evidències i garantir que el programari crític es comporta com s'espera. Aquests permeten fer un perfil exhaustiu de tots els esdeveniments que estan succeint internament en la placa [9].

El programari que s'ha utilitzat per a estudiar el comportament de la T2080 són dos programes de comportament controlat o benchmarks, un realitzarà lectures i l'altre escriptures [10].

En aquest treball només es tindrà en compte l'evidència empírica, ja que en altres articles s'ha demostrat que la definició d'alguns monitors en la documentació tècnica difereix dels resultats empírics en experiments concrets com el cas de NVIDIA Jetson i Xavier MPSoCs, i l'A53 en el Xilinx Zynq UltraScale+ MPSoC[11]. Per aquest motiu, construir un coneixement sòlid del hardware multi-core es fonamental per la seva certificació [12].

La forma d'estudiar el comportament de la placa es farà mitjançant la formulació d'hipòtesis, en elles es descriurà el comportament teòric esperat de la placa donats uns experiments. A través dels resultats generats pels PMCs, es confirmarà, rebutjarà o es modificarà la hipòtesi.

Posem per cas que tenim dos cores (C1 i C2) on els dos fan lectures a una regió de memòria compartida *vector*, si executem primer C1 i fa les lectures de *vector* en un temps $x$ i després fem que els dos cores llegeixin *vector* a la vegada, el temps d'execució del les dues execucions hauria de ser el mateix. A través dels PMCs de cada execució, es podrà validar, rebutjar o modificar la hipòtesi. En aquest cas només caldria mirar el comptador que mesura el temps d'execució de cada execució i comparar-los.

De forma molt simplificada, si es formula una hipòtesi conforme $2 + 2 = 4$ en l'àmbit teòric i s'introdueixen els dos operands en la màquina i dona un valor diferent de 4, això indica un comportament no esperat i cal investigar la seva causa. En el cas d'aquest treball, l'eina d'anàlisi seran els PMCs, els operands seran els programes i la màquina la T2080.

## III. Estat de l'art

Avui dia les aplicacions requereixen més recursos hardware i les arquitectures de les plataformes multi-core són cada cop més complexes. La coherència cau, donat el seu potencial per a millorar el rendiment entre l'intercanvi de dades, és un dels factors clau d'estudis recents[13], [14], [15], [16].

**A nivell acadèmic**, [13] proporciona una anàlisi del protocol MESI (Modified Exclusive Invalid Shared) i els seus inconvenients respecte a la previsibilitat del temps i demostra formes d'implementar dit protocol de tal forma que s'adapta millor als sistemes crítics.

Els autors a [14] van proposar invariants que garanteixen un comportament predicible en adoptar la coherència cau en sistemes de temps real i va demostrar l'aplicació d'aquestes invariants en proposar el protocol MSI (Modified Shared Invalid) predecible (PMSI).

Altres autors [15], suggereixen un protocol de coherència cau configurable basat en el temps dirigit a sistemes de criticalitat mixta.

En altres treballs [16], s'introdueix una solució que millora la latència de coherència sense degradar el rendiment del sistema. Aquests treballs estan disponibles a la xarxa amb l'enfocament d'aquest article que destaca la importància de la coherència cau en els sistemes crítics moderns.

En aquest article [17] s'analitza la coherència entre diferents clústers e6500 de la placa NxP T4240 i concloure que realment implementa MESIF en lloc de MESI tal com s'especifica al Technical Reference manual (TRM).

Els autors de [5], analitzen la coherència de la T2080 al nivell on es connecta l'L2 a la memòria principal, mitjançant com a elements el clúster i el DMA. En aquesta anàlisi identifiquen els esdeveniments que s'activen per les transicions entre estats de coherència, demostren que hi ha alguns PMCs amb descripcions ambigües o incompletes i detecten missatges de coherència inesperats en un sistema que conté només un sol clúster on mantenir la coherència.

En termes de hardware monitors, els autors a [11] reporten diferències entre els valors obtinguts de diversos monitors i la quantitat d'esdeveniments esperats segons la descripció d'esdeveniments de la documentació oficial del processador en els casos concrets en NVIDIA Jetson i Xavier MPSoCs, i A53 en Xilinx Zynq UltraScale+ MPSoC.

**A nivell industrial**, les errades en la documentació [18] capturen escenaris per l'arquitectura iMX6 de NXP en què alguns monitors de rendiment poden no comptar els esdeveniments amb precisió. Això reforça la necessitat d'evidenciar empíricament la correctesa dels PMCs.

## IV. Metodologia

El mètode que s'ha seguit per a poder assolir els objectius consisteix en l'aprofitament d'alguns principis de les metodologies agile i alguns de la metodologia SCRUM [19].

S'han fet tres reunions setmanals, dos amb el tutor del BSC i una amb el tutor de TFG, per a poder monitorar el progrés del projecte i inspeccionar l'estat d'aquest. Aquest feedback ha permès fer adaptacions en el mateix treball per tal d'arribar a bon port.

D'aquesta forma s'ha donat completa transparència sobre el progrés i la situació actual del projecte. Això ha estat fonamental per a poder detectar possibles errors o possibles millores en el desenvolupament del mateix treball.

En les mateixes reunions s'han fixat propòsits a curt termini que hauran de ser assolits de cara a la pròxima reunió. D'aquesta forma es pot dur a terme un monitoratge efectiu sobre l'estat del projecte i el progrés de l'objectiu a curt termini.

Finalment, s'han de realitzar iteracions de tot el que s'ha mencionat, ja que cada setmana hi ha hagut objectius a curt termini a obtenir i un monitoratge constant.

## V. Planificació

Per tal d'assolir l'objectiu del treball s'han proposat diversos passos a seguir els quals s'han fet de forma iterativa.

El primer pas ha estat la creació de l'entorn multicore en la IDE Codewarrior [20] i la creació dels programes de comportament conegut o benchmarks.

Seguidament, s'ha realitzat la formulació de la hipòtesi a verificar, en ella es descriu com s'hauria de comportar la placa T2080 d'acord amb la seva arquitectura i documentació tècnica [21]. Al principi s'ha començat amb hipòtesis senzilles, fàcilment comprovables i a

poc a poc s'ha passat a hipòtesis cada cop més complexes i difícils de comprovar.

Tenint en compte l'arquitectura i les hipòtesis prèviament formulades, s'ha fet el disseny dels experiments pertinents. Un cop dissenyats, s'ha fet la preparació dels experiments i s'han aplicat les modificacions necessàries per obtenir el funcionament esperat.

Els experiments estan formats per diferent micro-benchmarks els quals tenen múltiples variables per tal de poder adaptar-se de diverses maneres per ajustar-se al que es vol provar en cada experiment. Principalment, hem adaptat la mida de les estructures de dades, la separació entre accessos (STRIDE), la combinació dels programes (lectura / escriptura) i la compartició o no compartició de dades entre cores.

Els experiments s'han executat i d'allà s'han obtingut els resultats dels PMCs. A través de l'anàlisi dels resultats s'ha fet la resolució pertinent de la hipòtesi donada.

Pel fet que fa a la planificació, s'ha seguit la metodologia descrita en l'apartat anterior, on un objectiu a curt termini pot ser qualsevol dels passos descrits:

1) Formulació d'hipòtesi (com s'hauria de comportar / que hauria de passar teòricament)
2) Disseny dels experiments (creació dels micro-benchmarks)
3) Preparació dels experiments (adaptació dels micro-benchmarks per obtenir el funcionament desitjat)
4) Execució dels experiments (execució en el hardware)
5) Recol·lecció de resultats (extracció dels PMCs en brut i filtrat de les dades)
6) Interpretació dels resultats
7) Resolució
   a) Confirmació $\Longrightarrow$ Confirmem la hipòtesi
   b) Comportament anòmal $\Longrightarrow$ Indagació del origen del comportament (revisant codi assemblador, provant variants dels experiments, revisant documentació)
      - Tornar al pas 1

## VI. Entorn

Ens enfoquem en la NXP T2080 SoC [21] que implementa un clúster amb quatre cores d'arquitectura PowerPC e6500 [8] que està dissenyat pel seu ús en aviació [22].A la Figura 2, es veu el diagrama del blocs de la NXP T2080, la qual conté el clúster mencionat anteriorment connectat a CoreNet Coherence Fabric (CCF), que és la interconnexió que connecta amb tots els elements que utilitzen la memòria de la placa. La Figura 3, mostra amb més detall el clúster que té quatre cores, cada un amb una cau local (L1) per dades i un altre per instruccions, ambdós de 32kB, tots els cores comparteixen dades al següent nivell cau, a la L2 de 2MB i aquesta està connectada al CCF que interconnecta amb la memòria principal DDR. En tenir diferents nivells cau dins de l'arquitectura, ha d'implementar diferents protocols per mantenir la coherència. Per les L1, el sistema fa servir un protocol simple de Valid-Invalid on les escriptures són write-through, això significa que les memòries cau L1 poden tenir una mateixa línia cau valida si només fan servir Loads, però quan es realitza una escriptura en aquesta línia, s'invaliden totes les copies que estiguin a L1 per tots els cores (excepte el que està fent l'Store) i el nou valor s'escriu directament a la L2. La L2 implementa un protocol Modified-Exclusive-Shared-Invalid (MESI) basat en write-back, ja que pot compartir dades amb altres elements del sistema com podria ser el DMA.

## VII. Creació dels Benchmarks

En aquest apartat es procedeix a la creació dels programes de comportament conegut. Aquests programes anomenats benchmarks seran executats en l'arquitectura de la placa T2080 i a través dels comptadors de rendiment es podrà determinar el seu comportament en l'etapa d'experimentació. Es crearan dos programes de comportament conegut, un realitzarà lectures i l'altre escriptures.

Se seguiran les següents directrius al llarg del treball:

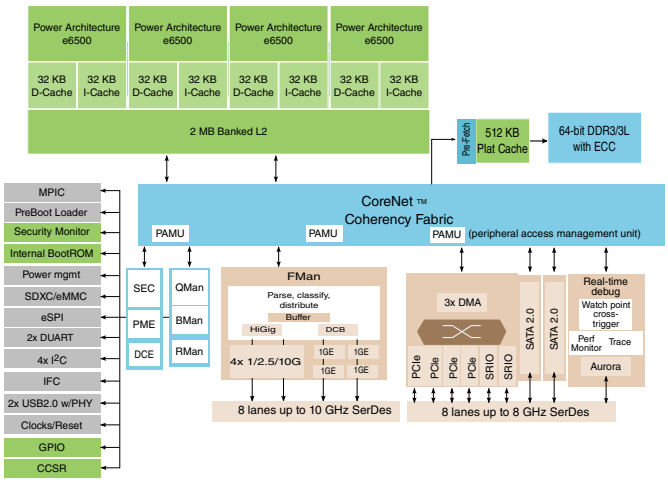



Figura 2: Diagrama de Blocs de la T2080

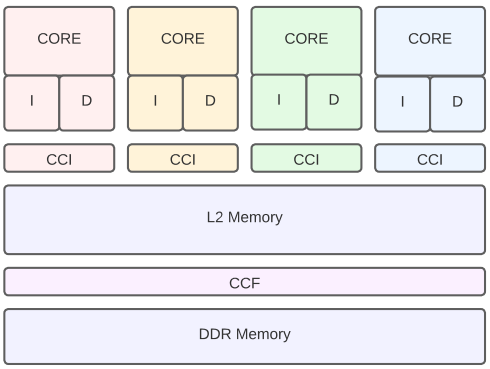



Figura 3: Diagrama de Blocs Simplificat de la T2080

- Totes les dades dels benchmarks seran de tipus Integer (4 bytes)
- La mida de l'estructura de dades serà de: VSIZE posicions * 4 bytes
- L'execució dels benchmarks es repeteix ITERS vegades, on ITERS = 100
- La compilació dels benchmarks es fa amb gcc 4.9.2 amb l'opció "O2"
- El nombre de posicions de memòria que se saltaran per a cada accés ho determina la variable STRIDE i a manera de conveni aquesta variable sempre s'expressa en Bytes.

El benchmark de lectura tindrà un Load i el benchmark d'escriptura un Store, en total faran VSIZE*ITERS Loads / Stores.

S'ha emprat l'opció "O2"d'optimització, ja que aquesta ens garanteix el comportament esperat del codi font vers el codi assemblador.

| Benchmark 1: leerVector (VSIZE, STRIDE) |
|---|
| Descripció: Llegeix elements de memòria consecutiva a distància STRIDE |
| Pseudocòdi (C) |
| `for(i = 0; i < VSIZE; i += STRIDE)`<br>`a+=dataset[i];` |
| Assembler |
| **lwz** r8,0(r9)<br>**addi** r9,r9,64 (STRIDE = 64 bytes)<br>**cmplw** cr7,r9,r7<br>**add** r10,r10,r8<br>**bne+** cr7,70 |

Taula II: Benchmark de lectures

| Benchmark 2: escribirVector (VSIZE, STRIDE) |
|---|
| Descripció: Escriu elements de memòria consecutiva a distància STRIDE |
| Pseudocòdi (C) |
| `for(i = 0; i < VSIZE; i += STRIDE)`<br>`dataset[i] = i;` |
| Assembler |
| `stw r10,0(r9)`<br>`addi r9,r9,64 (STRIDE = 64 bytes)`<br>`cmplw cr7,r9,r8`<br>`bne+ cr7,c0` |

Taula III: Benchmark d'escriptures

## VIII. Resultats Single Core

Aquesta etapa del treball consisteix a emprar hipòtesis o objectius on s'utilitzaran els benchmarks de comportament conegut en diversos experiments per a determinar el comportament de la T2080. Algunes Indicacions importants a tenir en compte són:

- Hi ha una fase d'inicialització que fa un warm-up de la memòria cau, això fa que tot accés als diferents nivells de la memòria cau facin hit en comptes de miss.
- Els experiments es divideixen en experiments single-core i multi-core. En ambdós casos es faran en memòria no compartida i en els experiments multicore també es faran experiments en memòria compartida.

### A. Mida del bloc cau

*1) Objectiu:* Mesurar i verificar la mida del bloc de memòria cau

*2) Paràmetres:* Benchmark: leerVector, STRIDE = 4, 8, 16, 64 B, VSIZE = 128K, Data Size = VSIZE*4

*3) Comptadors de rendiment:* Loads micro-ops completed i L1 cache misses

*4) Comportament esperat:* Donat que la memòria cau es divideix en blocs, i cada bloc agrupa dades adjacents, en accedir de manera seqüencial a les dades, on la mida de les dades és més gran que la mida de L1 ($524KB > 32KB$). El primer accés farà un miss, però tot accés a dins el bloc causarà hits.

En utilitzar dades de 4 bytes i sabent els misses per hit , es pot deduir la mida del bloc cau.

Per tant, si s'augmenta l'STRIDE o la distància d'accés entre dades, s'arribarà a un punt on hi haurà un miss per accés, això serà perquè a cada accés s'accedirà a un nou bloc cau i, com a resultat, STRIDE serà igual (o superior) al bloc cau.
$\frac{VSIZE*4Bytes}{BlockSize}$ on $BlockSize = \frac{VSIZE*4Bytes}{L1misses}$.

*5) Resultats:* A la Figura 4 es pot apreciar com el nombre de misses a L1 augmenta a mesura que l'STRIDE es fa més gran.

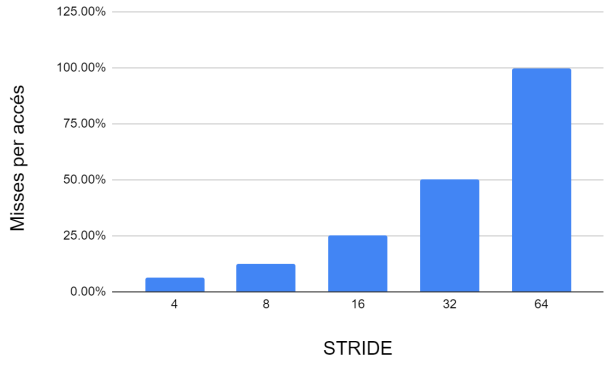



Figura 4: Misses per accés en funció d'STRIDE

*6) Conclusió:* Donat que es tenen 8K misses a L1 el $BlockSize = \frac{128K*4Bytes}{8192} = 64Bytes$. Tal com es pot veure a la Figura4 amb STRIDE = 4, 8, 16, 32 i 64 Bytes, es fan $BlockSize = \frac{1miss}{16accesses}, \frac{1miss}{8accesses}, \frac{1miss}{4accesses}, \frac{1miss}{2accesses} i \frac{1miss}{1access}$ respectivament.

Per tant, es compleix que STRIDE = mida del bloc = 64 Bytes, on s'accedeix cada cop a un nou bloc i el Nº de Loads = Nº d'accessos = L1 misses = 8192.

També es pot comprovar que $BlockSize = \frac{NLoads*STRIDE*4Bytes}{L1misses} = 64Bytes$. Per tant, es demostra que la mida del bloc cau és de 64 bytes.

### B. Mida de la memòria cau L1

*1) Objectiu:* Mesurar la mida de la memòria cau L1 a través de l'experimentació amb diferents mides de les dades, mesurant el nombre de misses a L1.

*2) Paràmetres:* Benchmark: leerVector, STRIDE = 64 B, VSIZE = 4K, 6K, 7.9K, 8K, 8.4K, 8.7K, 9.2K, 9.4K, 9.7K, 9.9K , Data Size = VSIZE*4

*3) Comptadors de rendiment:* Processor Cycles, Loads micro-ops completed i L1 cache misses

*4) Comportament esperat:* Si les dades són més petites que la mida de la L1, no hi haurà misses, ja que les dades caben a L1 i el temps d'accés serà el mateix.

En cas que les dades siguin més grans que la mida d'L1, hi haurà misses i aquests augmentaran a mesura que la mida de les dades augmenti, fins al punt de produir un miss per accés.

Els cicles per accés s'elevaran en sobrepassar la mida d'L1 i quan arribi al punt on cada accés produeix un miss, el temps d'accés es mantindrà.

*5) Resultats:* A la Figura 5 es pot apreciar l'augment de misses i a la Figura 6 del temps d'execució a mesura que creix el Data Size.

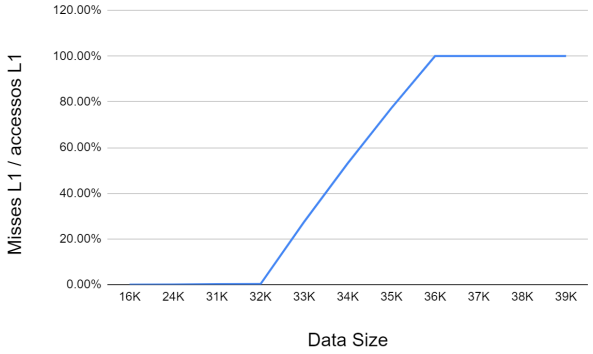



Figura 5: Misses en funció del Data Size

*6) Conclusió:* Tal com es pot veure en els resultats, el nombre de misses respecte al nombre d'accessos comença a fallar a L1 quan la mida de les dades sobrepassa els 32K Bytes.

El temps per accés es manté constant (8 cicles) fins als 32KB, quan es passa a una mida superior, el temps per accés augmenta fins al punt on hi ha un miss per accés on es manté (26 cicles).

Per tant, es verifica que la mida de la memòria cau L1 és de 32K.

### C. Mecanisme Write-through

*1) Objectiu:* Mesurar i verificar el mecanisme de coherència Write-through en el nivell L2 cau

*2) Paràmetres:* Benchmark: leerVector, escribirVector, STRIDE = 64 B, VSIZE= 4K, Data Size = VSIZE*4

*3) Comptadors de rendiment:* Loads/Stores micro-ops completed, L2 data accesses

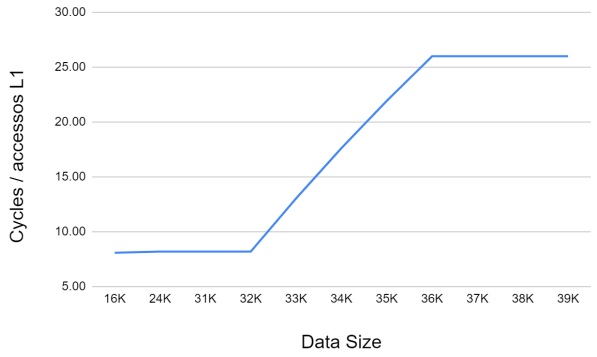



Figura 6: Temps de lectura en funció del Data Size

*4) Comportament esperat:* El mecanisme de coherència Write-through escriu a la memòria compartida L2 cada cop que un core realitza una escriptura en un bloc cau per mantenir els blocs actualitzats.

Es vol comprovar que quan es fan lectures, no es requereix cap mecanisme de coherència, ja que les dades no es modifiquen, però en cas de fer escriptures sí que es requereix un mecanisme per a mantenir els blocs cau actualitzats.

Per tant, s'espera que no hi hagi accessos a L2 en les lectures, ja que les dades no es modifiquen i caben a L1, però si s'espera accessos a L2 on Nº accessos a L2 = Nº Stores, pel fet que el write-through haurà d'accedir a L2 per actualitzar els blocs cau.

*5) Resultats:* A la Figura 7 es mostra el nombre d'accessos a L2 per part dels dos benchmnarks.

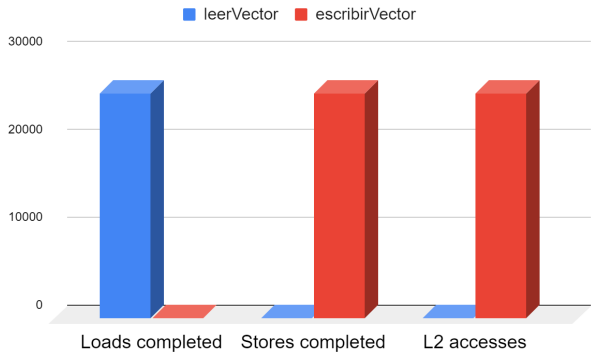



Figura 7: Accessos a L2 pel Write-Through

*6) Conclusió:* Donats els resultats, la ràtio d'accessos a L2 vers els Stores és d'1, hi ha un accés per Store amb el qual es compleix que Nº accessos a L2 = Nº Stores. Pel que fa als Loads, el nombre d'accessos vers els Loads s'aproxima a 0.

Per tant, es verifica que en el cas de lectures no es requereix cap mecanisme de coherència, però en el moment que hi ha escriptures sí que es requereixen mecanismes per mantenir coherència en el sistema.

### D. Execució Isolation dels Benchmarks

*1) Objectiu:* Mesurar el temps d'execució de cada benchmark amb diferents mides de dades

*2) Paràmetres:* Benchmark: leerVector, escribirVector, STRIDE = 64 B, VSIZE = 4K, 128K , Data Size = VSIZE*4

*3) Comptadors de rendiment:* Processor Cycles, Loads micro-ops completed, Stores micro-ops completed i L1 cache misses

| Data Size = 16K Bytes | leerVector | escribirVector |
|---|---|---|
| Processor cycles | 205083 | 179664 |
| Loads completed | 25604 | 4 |
| Store completed | 3 | 25602 |
| Data L1 cache misses | 0 | 25500 |

Taula IV: Execució en Isolation amb 16 KB

| Data Size = 524K Bytes | leerVector | escribirVector |
|---|---|---|
| Processor cycles | 21299506 | 5734864 |
| Loads completed | 819204 | 4 |
| Stores completed | 3 | 819202 |
| Data L1 cache misses | 819205 | 819100 |

Taula V: Execució en Isolation amb 524 KB

*4) Comportament esperat:* S'espera que quan les dades caben a L1 el temps d'accés de lectura / escriptura serà proper al temps de latència de les instruccions màquina i quan les dades siguin molt més grans que L1, el temps per accés de lectures / escriptures augmentarà tal com s'ha comprovat en el cas de les lectures a l'experiment Mida d'L1 VIII-B.

Segons el manual del processador e6500, la latència de les instruccions màquina dels benchmarks són les següents:

- lwz: 3 cicles
- addi: 1 cicle
- cmplw: 2 cicles (1 cicle quan els dos operands són iguals, bit EQ = 1)
- add: 1 cicle
- bne+: 1 cicle
- stw: 3 cicles

Per tant, quan la mida de les dades càpiga a L1, a leerVector s'espera una latència per iteració de 8 cicles i una latència de 3 cicles per lectura i en el cas de escribirVector s'espera una latència de 7 cicles per iteració i una latència de 3 cicles per escriptura. En cas que la mida de les $dades > mida\_L1$, el temps de lectura / escriptura augmentarà.

Comentari: Es pot apreciar que a leerVector hi ha una instrucció "add"de més, el qual suposa 1 cicle més que a escribirVector. Aquesta instrucció de suma és necessària en els loads, en cas que no hi fos, el compilador obviaria la resta i no es podria operar amb els loads perquè el codi en si no estaria fent res.

*5) Resultats:* En la Taula IV es pot extreure que amb una mida de dades de 16KB, el temps de les lectures és de $\frac{ProcessorCycles}{Loads} = \frac{205083}{25604} = 8cicles/iteració$. El temps per lectura serà de 3 cicles.

En el cas de les escriptures és de $\frac{ProcessorCycles}{Loads} = \frac{179664}{25602} = 7cicles/iteració$. El temps per escriptura serà de 3 cicles.

Amb una mida de dades de 524 KB (Taula V), el temps de les lectures és de $\frac{21299506}{819204} = 26cicles/iteració$. Per tant, el temps de lectura és de 21 cicles.

I en el cas de les escriptures és de $\frac{5734864}{819202} = 7cicles/iteració$. Per tant, el temps d'escriptura és de 3 cicles.

*6) Conclusió:* Quan les dades són de 16 KB, les lectures i les escriptures tenen un temps de 3 cicles, es compleix la latència de les instruccions màquina del codi assemblador.

Amb una mida de dades de 524 KB, el temps per lectura augmenta a 21 cicles i el temps d'escriptura es manté a 3 cicles.

En el cas de les lectures es compleix que quan s'augmenta la mida de les $dades > mida\_L1$ el temps de cada lectura augmenta x7 vegades respecte a quan la mida de les dades cap a L1.

Això és degut a la migració constant de blocs cau entre les memòries L1 i L2 tal i com indica el comptador de misses a L1 (819K misses). La mida de les dades resulta tan gran que s'han de fer accessos recurrents a L2 per a poder trobar els blocs cau requerits, aquesta recurrència a L2 fa que el temps de cada Load augmenti.

Per altra banda, en les escriptures quan s'augmenta la mida de les $dades > mida\_L1$, el temps d'escriptura es manté i no augmenta.

El motiu de per què no augmenta el temps de les escriptures tot i augmentar la mida de les dades és a causa de l'existència del "Store

| RR | 524 KB | 16 KB |
|---|---|---|
| **Memory** | - | - |
| **Processor cycles** | 22118700 | 205083 |
| **Load completed** | 819204 | 25604 |
| **Store completed** | 3 | 3 |
| **Data L1 cache misses** | 819205 | 3 |
| **BLINK request** | 0 | 0 |
| **L2 hits** | 1638413 | 9 |

Taula VI: Lectures amb 524 KB i 16 KB

| WW | 524 KB | 16 KB |
|---|---|---|
| **Memory** | - | - |
| **Processor cycles** | 5734864 | 179664 |
| **Load completed** | 4 | 4 |
| **Store completed** | 819202 | 25602 |
| **Data L1 cache misses** | 819100 | 25500 |
| **BLINK request** | 0 | 0 |
| **L2 hits** | 1638397 | 51196 |

Taula VII: Escriptures amb 524 KB i 16 KB

gather buffer", les escriptures a nivell de core s'han realitzat, però en realitat es guarden en un buffer per després fer les escriptures de forma superposada. D'aquesta forma es permet obtenir una latència de 3 cicles per escriptura accedint a L2 tal com s'ha observat en l'experiment.

## IX. Resultats Multi Core

En aquest punt es procedeix a mesurar i verificar els diferents estats de coherència cau "shared-invalid"en el nivell L2 i L1 de memòria cau. La part experimental es durà a terme amb dos processadors executats en paral·lel, on cada nucli conté un fil d'execució.

Quan els experiments multi nucli siguin de memòria compartida, significa que es comparteix la mateixa estructura de dades "dataset"entre nuclis, en el cas de memòria no compartida, cada nucli tindrà el seu propi "dataset".

Donat que en memòria no compartida cada nucli té les seves dades i operen independentment, això implica que no hi haurà invalidacions i, per tant, l'estat dels blocs cau romandran en l'estat "shared". Així i tot, s'utilitzaran els resultats en memòria no compartida per comparar-los amb els de memòria compartida.

Les execucions dels dos nuclis seran master-slave, quan el nucli master acabi l'execució també finalitzarà l'slave.

Per als experiments multicore, la nomenclatura per a leerVector serà R i en el cas de escribirVector serà W.

### *A. Verificar l'estat shared de coherència i mesurar el temps de lectura.*

*1) Objectiu:* Verificar i mesurar l'estat "shared" de la memòria cau L1 i L2 i mesurar el temps de lectura.

*2) Paràmetres:* Benchmark: leerVector, leerVector, STRIDE = 64 B, VSIZE= 128K, 4K , Data Size = VSIZE*4.

*3) Comptadors de rendiment:* Processor cycles, Loads completed, L1 misses, back-invalidations (BLINK) i L2 hits.

*4) Comportament esperat:* Donat que els dos nuclis fan lectures, les dades seran correctes en tot moment i no hi haurà invalidacions. Per tant, romandran en l'estat shared. El temps per Load no augmenta i s'espera que sigui de 3 cicles quan les dades siguin de 16 KB i de 21 cicles quan les dades siguin de 524 KB tal com s'ha demostrat a l'experiment VIII-D.

*5) Resultats:* En la Taula VI es poden apreciar els resultats extrets de l'execució de les lectures en paral·lel. En aquest cas els resultats en memòria compartida i no compartida són iguals.

*6) Conclusió:* Es confirma que les dades romanen en l'estat shared, això es pot corroborar amb el comptador de back-invalidations (BLINK) = 0. En cap moment s'han escrit / actualitzat els blocs cau.

En aquest cas emprar dos processadors no implica un increment del temps d'execució, això es pot corroborar de la següent forma:

- En el cas de 524 KB, $\frac{22118700}{819204} = 27 cicles/iteració$. El temps de lectura és de 22 cicles.
- En el cas de 16 KB, $\frac{205083}{25604} = 8 cicles/iteració$. El temps per lectura és de 3 cicles.

El temps per lectura no augmenta, dona els mateixos valors que en l'experiment VIII-D. En el cas de 524 KB el temps per lectura és de 22 cicles, 1 cicle més que en l'execució en isolation, això pot ser degut a que alguns Loads no s'han pogut superposar a l'inici de l'execució.

### *B. Verificar l'estat shared de coherència i mesurar el temps d'escriptura*

*1) Objectiu:* Verificar l'estat "shared" de la memòria cau L1 i L2 i mesurar el temps d'escriptura

*2) Paràmetres:* Benchmark: escribirVector, escribirVector, STRIDE = 64 B, VSIZE= 128K, 4K , Data Size = VSIZE*4

*3) Comptadors de rendiment:* Processor cycles, Stores completed, L1 misses, back-invalidations (BLINK) i L2 hits.

*4) Comportament esperat:* Els dos nuclis comparteixen i escriuen en les mateixes posicions de memòria, però al realitzar només escriptures (i cap lectura), això implica que no tenen cap còpia a la seva L1 privada i, per tant, no hi haurà cap invalidació de bloc cau. Com no s'ha invalidat cap bloc, aquests romandran en estat compartit.

El temps per escriptura s'espera que sigui de 3 cicles degut a la superposició de les escriptures com hem vist a l'experiment VIII-D.

*5) Resultats:* En la Taula VII es poden apreciar els resultats extrets de l'execució de les escriptures en paral·lel. En aquest cas els resultats en memòria compartida i no compartida són iguals.

*6) Conclusió:* Es confirma el comportament esperat, tot i tenir escriptures en les mateixes posicions de memòria el comptador BLINK = 0. Això és degut al fet que cap dels dos processadors té còpies dels blocs cau que s'estan escrivint.

El temps per escriptura es manté als 3 cicles, es pot confirmar de la següent forma:

- En el cas de 524 KB, $\frac{5734864}{819202} = 7 cicles/iteració$. El temps per escriptura és de 3 cicles.
- En el cas de 16 KB, $\frac{179664}{25602} = 7 cicles/iteració$. El temps per escriptura és de 3 cicles.

Es manté degut al "Store gather buffer" que permet acumular les escriptures i realitzar-les de forma superposada tal com s'ha vist a l'experiment VIII-D.

### *C. Verificar l'estat shared i invalid de coherència i el temps d'escriptura*

*1) Objectiu:* Verificar i mesurar l'estat "shared" i "invalid" de la memòria cau L1 i L2 i el temps d'escriptura.

*2) Paràmetres:* Benchmark: escribirVector, leerVector, STRIDE = 64 B, VSIZE= 128K, 4K , Data Size = VSIZE*4

*3) Comptadors de rendiment:* Processor cycles, Loads completed, Stores Completed, L1 misses, back-invalidations (BLINK) i L2 hits

*4) Comportament esperat:* S'espera que en memòria compartida hi hagi invalidacions, donat que s'està fent escriptures i lectures en les mateixes posicions de memòria i que en fer lectures es guarden còpies dels blocs cau, el nucli que executi les lectures tindrà invalidacions en el moment que es faci una lectura d'un bloc i aquest sigui escrit posteriorment.

*5) Resultats:* A la Taula VIII es poden apreciar els resultats extrets de l'execució de les escriptures i lectures en paral·lel. En aquest cas els resultats en memòria compartida i no compartida quan el Data Size és de 524KB no difereixen, la qual cosa no succeeix quan el Data Size és de 16 KB, en el següent apartat s'explicarà en detall.

| WR | 524 KB | 16 KB | 16 KB |
|---|---|---|---|
| **Memory** | - | Not Shared | Shared |
| **Processor cycles** | 5734864 | 179664 | 179664 |
| **Load completed** | 4 | 4 | 4 |
| **Store completed** | 819202 | 25602 | 25602 |
| **Data L1 cache misses** | 819100 | 25500 | 25500 |
| **BLINK request** | 0 | 0 | 0 |
| **L2 hits** | 1029588 | 25603 | 32152 |

Taula VIII: Escriptures i Lectures amb 524 KB i 16 KB

| RW | 524 KB | 524 KB | 16 KB | 16 KB |
|---|---|---|---|---|
| **Memory** | Not Shared | Shared | Shared | Not Shared |
| **Processor cycles** | 22121258 | 22323552 | 699143 | 205084 |
| **Load completed** | 819204 | 819204 | 25604 | 25604 |
| **Store completed** | 3 | 3 | 3 | 3 |
| **Data L1 cache misses** | 819205 | 819205 | 25605 | 3 |
| **BLINK request** | 0 | 818470 | 25594 | 0 |
| **L2 hits** | 3979090 | 4007992 | 125188 | 29212 |

Taula IX: Lectures i Escriptures amb 524 KB i 16 KB

*6) Conclusió:* Es pot apreciar que en cap cas BLINK = 0. Això és degut al fet que s'està mirant per la banda del W, per afirmar o negar invalidacions s'utilitzarà el comptador d'L2 hits = Nº accessos a L2, a més dels resultats obtinguts en memòria no compartida.

En el cas de 524 KB els resultats són els mateixos, però per motius diferents.

En memòria no compartida simplement els $L2hits = 819K + \frac{1}{4} * 819K = 102Khits$, on els 819K hits són del W i els hits restants de R pel fet que les escriptures són 4 cops més ràpides i al finalitzar només s'han pogut fer $\frac{1}{4}$ dels accessos a L2 per part de les lectures.

En memòria compartida, el raonament és que poden haver invalidacions, però donat que la mida de les dades és molt gran i es genera un miss a L1 per accés, es dona la situació que les invalidacions queden eclipsades per la constant migració de blocs cau.

En el cas de 16 KB, en memòria no compartida els L2 hits = Nº Stores, per tant, no hi ha invalidacions. En memòria compartida $L2hits > NStores$, en aquest cas hi ha els 25K hits de les escriptures i 6.5K hits de les invalidacions de les lectures. El temps per escriptura es manté a 3 cicles en tots els casos, veure conclusions de l'experiment IX-B.

### *D. Verificar l'estat shared i invalid de coherència i el temps per lectura*

*1) Objectiu:* Verificar i mesurar l'estat "shared" de la memòria cau L1 i L2 i el temps per lectura

*2) Paràmetres:* Benchmark: leerVector, escribirVector, STRIDE = 64 B, VSIZE= 128K, 4K , Data Size = VSIZE*4

*3) Comptadors de rendiment:* Processor cycles, Loads completed, Stores Completed, L1 misses, back-invalidations (BLINK) i L2 hits

*4) Comportament esperat:* En el cas de memòria compartida, en realitzar una lectura i després una escriptura en la mateixa posició de memòria, la còpia del bloc cau que ha fet la lectura s'invalida ja que el bloc s'ha actualitzat. Tots els blocs cau obsolets a L1 seran invalidats i romandran en l'estat ïnvalid"fins a rebre la còpia actualitzada del bloc cau. La resta de blocs cau continuen en l'estat "shared".

S'espera que quan la mida de les dades sigui de 524 KB, hi haurà un miss per accés amb el qual les invalidacions seran eclipsades (en l'àmbit de rendiment) per la migració de blocs entre memòries, com hem vist a l'experiment IX-C. En el cas de 16 KB s'espera una latència més elevada a causa de les invalidacions entre la versió de memòria compartida i la de memòria no compartida.

*5) Resultats:* A la Taula IX es poden apreciar els resultats extrets de l'execució de les lectures i escriptures en paral·lel.

*6) Conclusió:* Tal com es pot veure, en el cas de 524 KB les invalidacions BLINK = 818K no representen un increment important en el temps d'execució pel fet que hi ha un miss per accés tal com passava en l'experiment IX-C. El temps de lectura és de 22 cicles.

En el cas de 16 KB no hi ha misses a L1 per part dels loads amb el qual es pot observar com les invalidacions BLINK = 26K provoquen que el temps per lectura sigui més lenta vers la versió not shared. Concretament, les invalidacions condicionen un temps per iteració de $\frac{699143}{25604} = 27cicles/iteració$. Amb un temps de lectura de 22 cicles.

I en el cas sense invalidacions, el temps per iteració és de $\frac{205084}{25604} = 8cicles/iteració$. Amb un temps de lectura de 3 cicles.

Per tant, en aquest cas les invalidacions produeixen que el temps de lectura sigui $\frac{22}{3} = 7.3x$ vegades més lent.

## X. Conclusions

En l'experimentació single-core s'ha procedit a realitzar diversos experiments per formar la base sobre la qual es treballarà més endavant.

En el primer experiment s'ha pogut comprovar que l'accés a una posició de memòria a una distància de 64 bytes, s'accedeix a un nou bloc cau on es compleix que $1accés = 1miss$ a L1, per tant, la mida del bloc cau és de 64 Bytes.

En la mida de la memòria cau L1 s'ha comprovat l'augment de misses i del temps d'execució quan les dades són més grans que 32KB, on cada lectura té una duració de 3 cicles en cas que $DataSize \leq 32KB$ i en quant les dades augmenten de mida on $DataSize > 32KB$, s'assoleix un temps de 21 cicles per load. Aquest augment dels misses i del temps d'execució confirmen que la memòria cau L1 és de 32 KB.

Pel que fa a la coherència, quan es realitzen lectures tots els blocs cau són correctes i no es requereix cap mecanisme de coherència. Així i tot, això canvia quan hi ha escriptures, en aquest cas el nivell de coherència es troba a la memòria cau L2 i utilitza el mecanisme write-through per actualitzar els blocs cau que puguin ser modificats a la memòria cau L2.

Les execucions single-core dels dos benchmarks, quan el $DataSize > 32KB$ el temps de cada Load s'incrementa fins els 21 cicles i els Stores es mantenen a 3 cicles. En el cas dels Loads, hi ha un miss per accés i la migració de blocs cau és constant entre les memòries L1 i L2, la qual cosa alenteix l'execució. En el cas de les escriptures, el temps es manté a causa del "Store gather buffer", les escriptures s'acumulen en un buffer i aquestes són realitzades de forma superposada.

Un cop verificat el correcte funcionament dels monitors de rendiment i amb nocions del funcionament intern de la T2080, es procedeix a fer l'experimentació de coherència amb dos processadors que s'executen en paral·lel.

L'execució de lectures fa que els blocs cau estiguin en l'estat shared donat no hi ha escriptures.

En el cas d'escriptures, les dades es mantenen en l'estat shared tot i escriure en les mateixes posicions de memòria perquè cap dels processadors tenen còpies dels blocs cau que estan modificant.

En el cas d'escriptures i lectures, si hi ha invalidacions, però no es veuen reflectides. En aquest cas per demostrar l'existència d'invalidacions s'utilitza els accessos a la memòria L2. El cas on es veu reflectit és quan el Data Size = 16 KB. En memòria no compartida L2 hits = Nº Stores, per tant, no hi ha invalidacions. En memòria compartida $L2hits > NStores$, en aquest cas hi ha els 25K hits de les escriptures i 6.5K hits de les invalidacions de les lectures.

En les lectures i escriptures, en el cas que el Data Size = 524 KB, les invalidacions no representen un increment important en el temps d'execució, ja que hi ha un miss per accés on el temps de lectura és de 22 cicles i això és el que més alenteix l'execució. En el cas que el Data Size = 16 KB, no hi ha misses a L1 per part dels loads amb el qual es pot observar com en la versió compartida, les invalidacions provoquen que el temps per lectura sigui de 22 cicles, 7.3 cops més lent vers la versió no compartida.

## XI. Agraïments

En aquest apartat vull agrair el temps, directrius i dedicació del professor Juan Carlos Moure del departament d'Arquitectura de Computadors i Sistemes Operatius (CAOS) en aquest Treball de Fi de Grau.

Per últim, vull agrair a en Fransico Cazorla per la oportunitat de fer aquest TFG en col·laboració amb el Barcelona Supercomputing Center (BSC) i al meu supervisor Roger Pujol pel seu suport al llarg del treball.

## Referències

[1] W. B. III, "Avionics industry advances toward dal a multicore adoption," Aviation Today, dec 2019. [Online]. Available: https://interactive.aviationtoday.com/avionicsmagazine/december-2019-january-2020/avionics-industry-advances-toward-dal-a-multicore-adoption/_fragment.html

[2] P. S. Vadgaonkar and U. Janardhan, "Do-254/ed-80 - an application guidelines to redesign/re-engineering airborne electronic hardware," in *SAE 2016 Aerospace Systems and Technology Conference*. SAE International, sep 2016. [Online]. Available: https://doi.org/10.4271/2016-01-2039

[3] Military Aerospace Electronics, "What is safety-certifiable avionics hardware that meets design assurance levels (dal)?" Military Aerospace Aelectronics, nov 2016. [Online]. Available: https://www.militaryaerospace.com/computers/article/16714656/what-is-safetycertifiable-avionics-hardware-that-meets-design-assurance-levels-dal

[4] Mistral Solutions, "Airborne electronics – airworthiness regulations and safety requirements," Mistral Solutions, 2020. [Online]. Available: https://www.mistralsolutions.com/blog/airborne-electronics/

[5] R. Pujol, H. Tabani, J. Abella, M. Hassan, and F. J. Cazorla, "Empirical evidence for mpsocs in critical systems: The case of nxp's T2080 cache coherence," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, 2021, pp. 1162–1165. [Online]. Available: https://doi.org/10.23919/DATE51398.2021.9474078

[6] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P. G. Zaykov, Z. Petrov, B. Böddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Cassé, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quiñones, M. Panic, J. Abella, F. J. Cazorla, S. Uhrig, M. Rohde, and A. Pyka, "parmerasa - multi-core execution of parallelised hard real-time applications supporting analysability," in *2013 Euromicro Conference on Digital System Design, DSD 2013, Los Alamitos, CA, USA, September 4-6, 2013*. IEEE Computer Society, 2013, pp. 363–370. [Online]. Available: https://doi.org/10.1109/DSD.2013.46

[7] D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2011. [Online]. Available: https://doi.org/10.2200/S00346ED1V01Y201104CAC016

[8] *e6500 Core Reference Manual*, https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf, Freescale Semiconductors, 6 2014, rev. 0.

[9] MASPA Technologies, "Hardware analysis services," 2020. [Online]. Available: https://maspatechnologies.com/hardware-analysis-services-and-products/

[10] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*, 2012, pp. 91–101. [Online]. Available: https://doi.org/10.1109/ECRTS.2012.31

[11] J. Barrera, L. Kosmidis, H. Tabani, E. Mezzetti, J. Abella, M. Fernández, G. Bernat, and F. J. Cazorla, "On the reliability of hardware event monitors in mpsocs for critical domains," in *SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020*, 2020, pp. 580–589. [Online]. Available: https://doi.org/10.1145/3341105.3373955

[12] I. Agirre, J. Abella, M. Azkarate-askasua, and F. J. Cazorla, "On the tailoring of CAST-32A certification guidance to real COTS multicore architectures," in *12th IEEE International Symposium on Industrial Embedded Systems, SIES 2017, Toulouse, France, June 14-16, 2017*. IEEE, 2017, pp. 1–8. [Online]. Available: https://doi.org/10.1109/SIES.2017.7993376

[13] S. Uhrig, L. Tadros, and A. Pyka, "Mesi-based cache coherence for hard real-time multicore systems," in *Architecture of Computing Systems - ARCS 2015 - 28th International Conference, Porto, Portugal, March 24-27, 2015, Proceedings*, 2015, pp. 212–223. [Online]. Available: https://doi.org/10.1007/978-3-319-16086-3_17

[14] M. Hassan, A. M. Kaushik, and H. D. Patel, "Predictable cache coherence for multi-core real-time systems," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2017, Pittsburg, PA, USA, April 18-21, 2017*, 2017, pp. 235–246. [Online]. Available: https://doi.org/10.1109/RTAS.2017.13

[15] N. Sritharan, A. M. Kaushik, M. Hassan, and H. D. Patel, "Enabling predictable, simultaneous and coherent data sharing in mixed criticality systems," in *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, 2019, pp. 433–445. [Online]. Available: https://doi.org/10.1109/RTSS46320.2019.00045

[16] M. Hassan, "Discriminative coherence: Balancing performance and latency bounds in data-sharing multi-core real-time systems," in *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, July 7-10, 2020, Virtual Conference*, 2020, pp. 16:1–16:24. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2020.16

[17] N. Sensfelder, J. Brunel, and C. Pagetti, "On how to identify cache coherence: Case of the NXP qoriq T4240," in *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, July 7-10, 2020, Virtual Conference*, 2020, pp. 13:1–13:22. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2020.13

[18] N. Semiconductors, "Chip Errata for the i.MX 6SLL. Document Number: IMX6SLLCE," 2019.

[19] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, ser. Agile Software Development. Prentice Hall, 2002. [Online]. Available: https://books.google.es/books?id=BpFYAAAAYAAJ

[20] *CodeWarrior Development Studio for Microcontrollers V10.x ColdFire Build Tools Reference Manual*, https://www.nxp.com/docs/en/reference-manual/CWMCUCFCMPREF.pdf, Freescale Semiconductors, 02 2014, rev. 10.6.

[21] *T2080 Product Brief*, https://www.nxp.com/docs/en/product-brief/T2080PB.pdf, Freescale Semiconductors, 4 2014, rev. 0.

[22] D. Radack, H. Tiedeman Jr, and P. Parkinson, "Civil certification of multi-core processing systems in commercial avionics," February 2019.