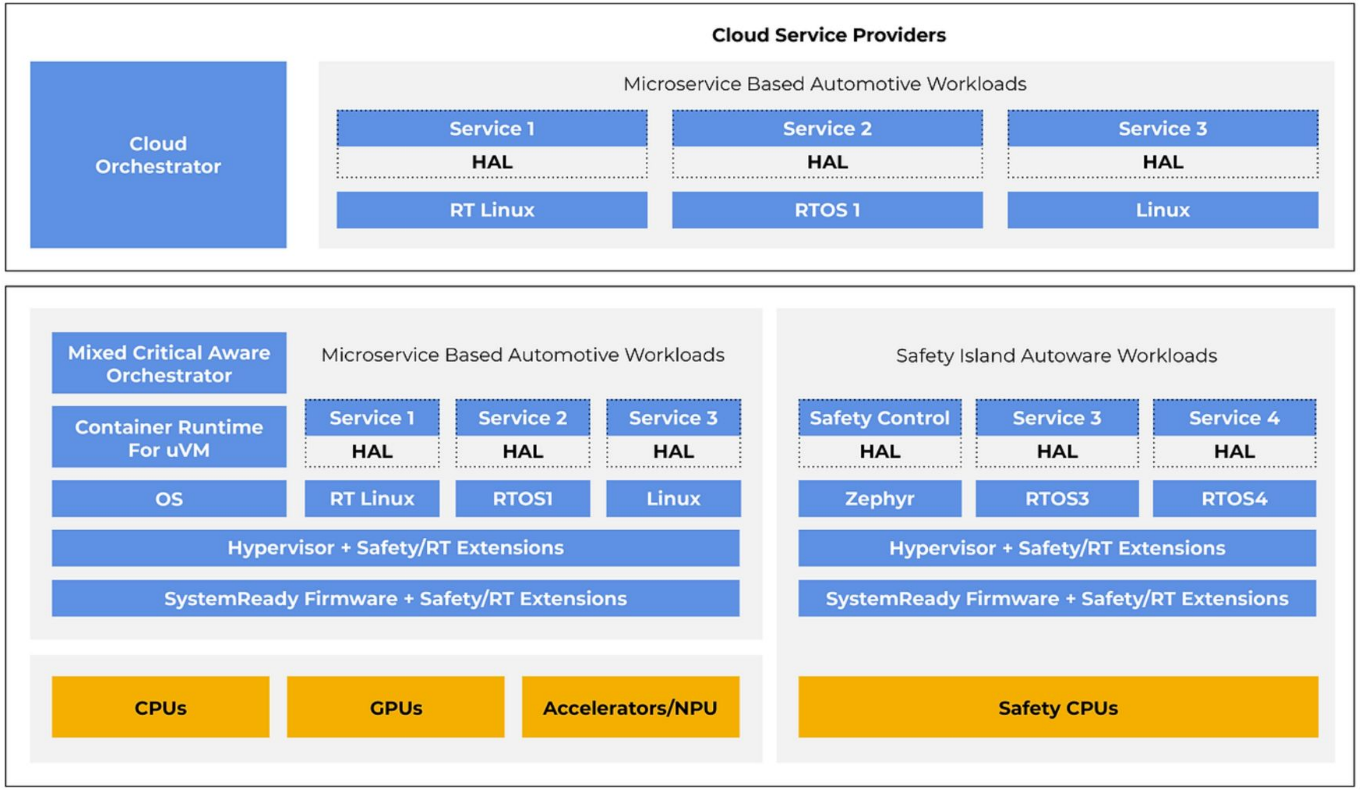
CA2022 期末考古解析

1. Complex Hardware-Software Interactions


Cloud Service Providers
Cloud Orchestrator
Microservice Based Automotive Workloads
Service 1
HAL
RT Linux
Service 2
HAL
RTOS 1
Service 3
HAL
Linux
Mixed Critical Aware Orchestrator
Container Runtime For uVM
OS
Microservice Based Automotive Workloads
Service 1
HAL
RT Linux
Service 2
HAL
RTOS1
Service 3
HAL
Linux
Hypervisor + Safety/RT Extensions
SystemReady Firmware + Safety/RT Extensions
CPUs
GPUs
Accelerators/NPU
Safety Island Autoware Workloads
Safety Control
HAL
Zephyr
Service 3
HAL
RTOS3
Service 4
HAL
RTOS4
Hypervisor + Safety/RT Extensions
SystemReady Firmware + Safety/RT Extensions
Safety CPUs

(a) It would not be safe if the safety CPU fails. Please describe how to realize safety CPU with multiprocessors in this case.

可以透過以下幾種機制在多處理器環境下來實現safety CPU.

異常偵測 : 多個處理器可以執行相同工作，結果不同時，即代表發生異常。

互相備緩機制 : 處理器之間可以互相備緩，當某個處理器 有異常時，正常的處理器去取代異常處理器在執行的工作。

Triple Modular Redundancy (TMR): 讓 3 個相同的處理器來執行相同的工作。如果1 個處理器異常，因為仍有 2 個處理器結果為正確的，透過投票機制(即 2 vs 1) ，仍能得出正確結果。

（這題沒有標準答案，只要寫得合理應可得分）

(b) Tt is hard to guarantee real-time processing for an application in a regular operating system especially when there are many concurrent tasks within an OS. Can you utilize multiprocessors to improve real-time processing? How?

1. 在一般的 OS 環境下，加入 realtime priority 機制，讓 real time task 有絕對的優先權。而相較於單處理器環境，在多處理器環境下，real time tasks 可以被分配到不同的處理器，避免 real time task 之間彼此競爭處理器資源，讓執行時間更能夠被預測，確保 real time task 能在時間內完成。

2. 獨立出幾個處理器專門執行real time task，並且在對在這幾個處理器上執行的real time task 採用 real time 的排程演算法。確保這些real time task 的執行時間能被預測，反應更即時。

以上二種答案均可。

(c) Please explain why and how hypervisor may help improve the interactions between the vehicle and the cloud service providers.

從題目說明可以看到，車上系統會下載cloud provider 的程式下來執行。因為執行從網路下載的程式本身是一件不太安全的事，而hypervisor 提供了更好的隔離的功能，故在提供 cloud provider 互動時的安全性可以被顧及。

又因為 hypervisor 允許從 cloud provider 下載的這些程式/服務在車上執行，hypervisor 可以在確保安全的情況下，建立快速通道（例如用shared memory），使得車載的系統與cloud provider 的應用程式，可以方便迅速地交換資料。

（這題沒有標準答案，只要寫得合理應可得分）

(d) Please explain why and how hypervisor may help improve the safety and real-time processing in this case.

Hypervisor 提供 fault-isolation 機制：不同 VM 間，即使一個 VM 壞了，或運作異常，也不會影響到其他 VM 的運作。

此外 Hypervisor 也可以採靜態資源分配，將幾個處理器配給一個或多個獨立VM 專門跑 Real Time OS 及處理 Real Time task，透過靜態資源分配，可以少掉VM Scheduler 的開銷，使得這些 Real Time task 反應更即時，更能預測他們的執行時間。
(例如：Bao Hypervisor)

(e)

CPU A run 1GHz & execute 2 FP/s

CPU B run 250MHz & execute 16 FP/s

$\rightarrow$ A : 1 G × 2 FP/s = 2 GFLOPs /sec

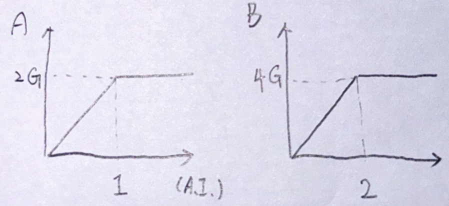
B : 250 M × 16 FP/s = 4 GFLOPs /sec

(f)

GFLOPs/s = A.I. × $\underline{\text{memory bandwidth}}$ = 2 GB/s

A $\rightarrow$ 2 GFLOPs/s $\rightarrow$ A.I. = 1

B $\rightarrow$ 4 GFLOPs/s $\rightarrow$ A.I. = 2



(G)

```
int Count = 0
  for (i = 0~1024){
    for (j = 0~1024){
      if(X[i][j] > Y[i][j]){
        count ++;
      }
    }
  }
```

$\rightarrow$ memory access :

X[i][j] & Y[i][j] for all i,j

$\rightarrow$ 2 × 1024 × 1024

F.P. operation :

1 comparison $\rightarrow$ 1024 × 1024

$\rightarrow$ A.I = 0.5

(h)

因為比大小是一次性的

,沒有利用到 temporal locality

所以 cache might not be efficient at all.

(i)

A, B 皆在 memory bound $\because$ A.I. = 0.5

$\rightarrow \frac{1\,M}{250\,M}$ (s) = 4 ms.

(j)

```
for (i = 0~n){
  for (j = 0~n){
    for (k = 0~n){
      C[i][j] += a[i][k] * b[k][j];
    }
  }
}
```

(k)

因為 access 是跳動的 $\rightarrow$ 沒有利用到 spatial locality

$\rightarrow$ not efficient

(L) in k's for loop :

load 3 FP & calculate 2 FP (mult & add)

$\rightarrow$ A.I = $\frac{2}{3}$, A, B 皆在 memory bound

$\rightarrow \frac{2 \times n^2}{500\,M \times \frac{2}{3}} = \frac{2 \times n^2}{333.33\,M}$ sec (n 未知)

(M)

arithmetic intensity $\uparrow$

by hit rate $\uparrow$

& memory access count $\downarrow$

(n)

A 做 memory bound

B 做 computation bound

(有 SIMD)

3. Stream Processing

(o) We consider reading the two matrices X, Y from memory.
(In this workload, most of the memory access is for this)

Suppose the elements in the matrices are stored sequentially in memory.
When we read an element, if it's not in the cache (cache miss),
the whole memory block is brought to the cache block. Since the elements
are stored sequentially, the following elements are brought together to the cache.

A 32-byte cache block can store 8 4-byte floating points.

⇒ 1 cache miss results in 7 cache hits

⇒ miss rate = $\frac{1}{8}$ = 12.5%

(P) ① larger cache block size ⇒ lower miss rate

② Since every elements is used once, all cache misses are "compulsory" misses

(q) block size = 16 bytes, miss rate = $\frac{1}{4}$ = 25%

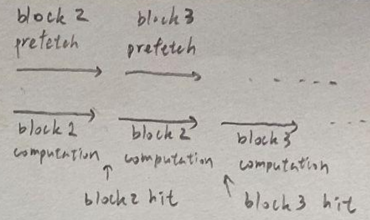
block size = 64 bytes, miss rate = $\frac{1}{16}$ = 6.25%

block size = 128 bytes, miss rate = $\frac{1}{32}$ = 3.13%

"Spatial locality"

(r)

miss rate = 0



(s) Each processor runs this program (SPMD)
(Suppose we have N processors)

```
counts [N] = 0          // shared array
pid = get_processor_id()
n_task = 1024*1024 / N        equally distribute
                              the workload for SMP
for (i = n_task * pid ; i < n_task * (pid + 1); i++ )
    r = (int) i / 1024
    c = i % 1024
    if ( X[r][c] >= Y [r][c] )
        counts[ pid ] += 1

if (pid == 0)
    result = sum (counts)
    print ( result)
```

(t)

GPU has hundreds of arithmetic units.
It can execute hundreds of threads parallelly.