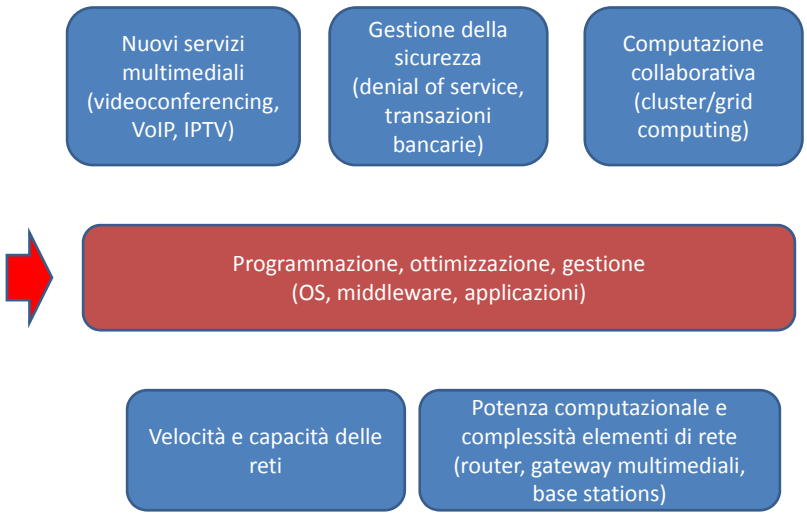
# Distributed Embedded Systems

Andrea Acquaviva

## Scopo del Corso



# Contenuti del Corso

- Programmazione parallela per sistemi embedded
- Il corso è organizzato in vari moduli, in genere uno per lezione
- Sommario dei contenuti
  - Applicazioni (videoconferenza, VoIP, encryption, packet scheduling, transcodifica video, etc)
  - Modelli computazionali (kernel computazionali, control e data-flow oriented applications, task graph)
  - Hardware (piattaforme multiprocessore, memorie, interconnection network, primitive di comunicazione, affidabilità)
  - Modelli di programmazione (message passing, shared memory, threads)
  - Software di sistema (compilatori, autotuners, OS, virtual machines)
  - Metriche e ottimizzazione (allocazione e bilanciamento del carico e della comunicazione)

# Introduction

- In 2005 Intel announced that its high performance microprocessor would rely on multiple processors or cores
  - OK for multiprogrammed workload, but what for single tasks?
  - Programmability?

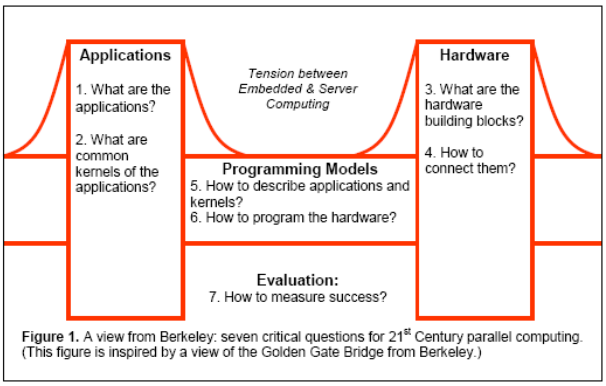



**Figure 1.** A view from Berkeley: seven critical questions for 21st Century parallel computing. (This figure is inspired by a view of the Golden Gate Bridge from Berkeley.)

# INTRO I

Application Scenario

## Embedded vs Server Computing

- They have more in common than in the past
  - Power
  - Hardware utilization (cost)
  - Software reuse (limit hand tuning)
  - Network connection (protection, resource sharing and scheduling)
  - Real time requirements

# Motivation (1/3)

*1. Old CW: Power is free, but transistors are expensive.*

*New CW is the "Power wall": Power is expensive, but transistors are "free". That* is, we can put more transistors on a chip than we have the power to turn on.

2. *Old CW: If you worry about power, the only concern is dynamic power.*

*New CW: For desktops and servers, static power due to leakage can be 40% of* total power.

3. *Old CW: Monolithic uniprocessors in silicon are reliable internally, with errors* occurring only at the pins.

*New CW: As chips drop below 65 nm feature sizes, they will have high soft and* hard error rates. [Borkar 2005] [Mukherjee et al 2005]

4. *Old CW: By building upon prior successes, we can continue to raise the level of* abstraction and hence the size of hardware designs.

*New CW: Wire delay, noise, cross coupling (capacitive and inductive),* manufacturing variability, reliability (see above), clock jitter, design validation, and so on conspire to stretch the development time and cost of large designs at 65 nm or smaller feature sizes.

5. *Old CW: Researchers demonstrate new architecture ideas by building chips.*

*New CW: The cost of masks at 65 nm feature size, the cost of Electronic* Computer Aided Design software to design such chips, and the cost of design for GHz clock rates means researchers can no longer build believable prototypes. Thus, an alternative approach to evaluating architectures must be developed.

# Motivation (2/3)

6. *Old CW: Performance improvements yield both lower latency and higher* bandwidth.

*New CW: Across many technologies, bandwidth improves by at least the square* of the improvement in latency. [Patterson 2004]

7. *Old CW: Multiply is slow, but load and store is fast.*

*New CW is the "Memory wall" [Wulf and McKee 1995]: Load and store is slow,* but multiply is fast. Modern microprocessors can take 200 clocks to access Dynamic Random Access Memory (DRAM), but even floating point multiplies may take only four clock cycles.

*8. Old CW: We can reveal more instruction-level parallelism (ILP) via compilers* and architecture innovation. Examples from the past include branch prediction, out-of-order execution, speculation, and Very Long Instruction Word systems.

*New CW is the "ILP wall": There are diminishing returns on finding more ILP.* [Hennessy and Patterson 2007]

9. *Old CW: Uniprocessor performance doubles every 18 months.*

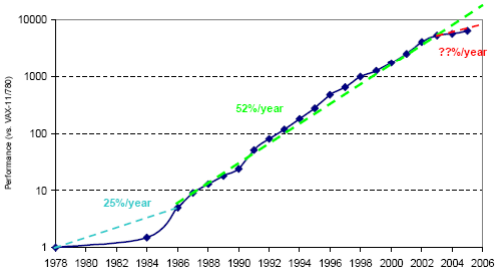
*New CW is Power Wall + Memory Wall + ILP Wall = Brick Wall. Figure 2 plots* processor performance for almost 30 years. In 2006, performance is a factor of three below the traditional doubling every 18 months that we enjoyed between 1986 and 2002. The doubling of uniprocessor performance may now take 5 years.

# Motivation (3/3)

10. *Old CW: Don't bother parallelizing your application, as you can just wait a little* while and run it on a much faster sequential computer.
*New CW: It will be a very long wait for a faster sequential computer (see above).*

11. *Old CW: Increasing clock frequency is the primary method of improving* processor performance.
*New CW: Increasing parallelism is the primary method of improving processor* performance.

12. *Old CW: Less than linear scaling for a multiprocessor application is failure.*
*New CW: Given the switch to parallel computing, any speedup via parallelism is a* success.




# Parallel Computing Applications

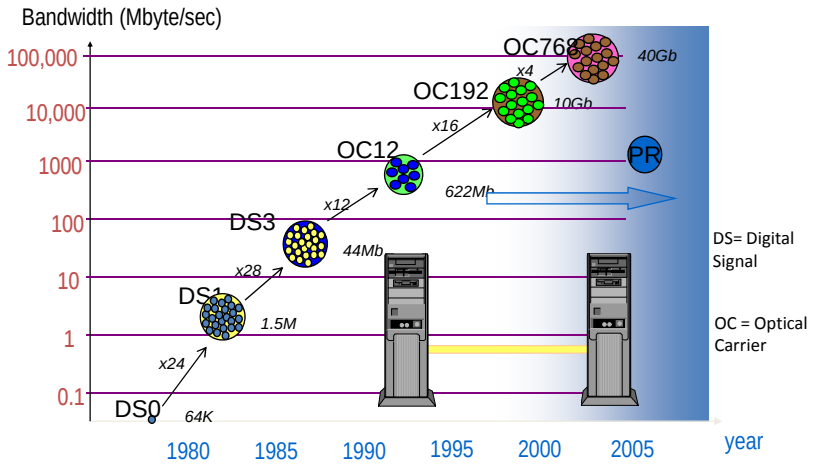
Parallel computing

- General purpose (PCs)
- Network processors (routers, media gateway, base stations)
- Media servers (url balancing)
- Scientific and entertainment computing (physical event modelling, video rendering)
- Mobile terminals (palmtops, smart-phones)

# Evolving Network Scenario (1/2)

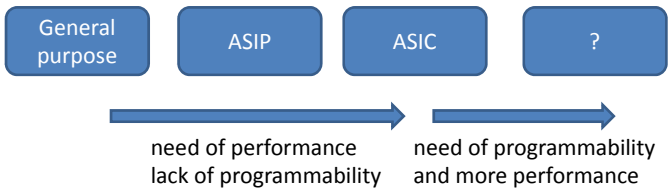
- Network speed is increasing
  - High speed routers: OC-192 (10Gb/s) or OC-768 ($40Gb/s)
- Network applications requirements are increasing and more functionalities are needed along with packet processing
  - Packet content processing
- New services:
  - Packet classification, QoS support, real-time constraints, traffic shaping, intrusion detection, security support, active networks
  - VoIP gateways, video transcoding

# Evolving Network Scenario (2/2)

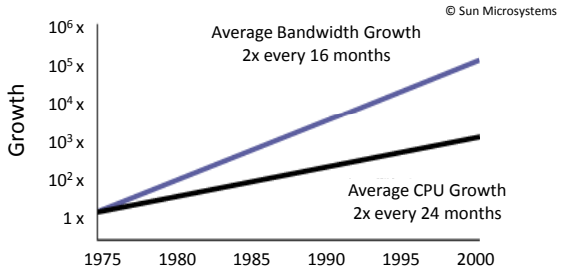


# Impact on Network Equipment

- Increasing computational effort as applications are more sophisticated
  - Flexibility to support different and evolving network protocols (e.g. security applications)
  - Processing at line rates
- Current programmable network processors such as the Intel IXP2800 target low performance (100Mbs to 10Gbps)
- Evolution




# High Performance & Flexibility




- Growth in networking bandwidth has outstripped the advances in computing
- ASIC-based high-performance networking devices are inflexible and impede introduction of new services
- Need for ability to rapidly develop and deploy new services into existing networks
- Need for programmability with high performance in the data-plane

14

# Network Application Complexity (1/2)

| | Network Application | Complexity Insts per byte | |
|---|---|---|---|
| Header Processing | Deficit Round Robin | 4.1 | |
| | IP Header Fragmentation | 7.7 | |
| | Radix Tree Routing | 2.1 | |
| | TCP Filter Matching | 10.3 | |
| Payload Processing | | Encoding | Decoding |
| | Encryption | 104 | 104 |
| | JPEG Transcoding | 81 | 60 |
| | Reed-Solomon FEC | 603 | 1052 |
| | Lempel-Ziv Compression | 226 | 35 |

[Wolf@UMass]

**Single CPU example**:

- Network speed: 10Gbps
- Packet arrival rate: ~50ns
- CPU frequency: 2.4GHz
- Clock Period: 0.41ns
- Insts per clock: 4
- Insts per packet: ~500
  - Line-rate processing
- Insts per byte: ~8

15

# Network Application Complexity (2/2)

- This allows an average of 500 instructions that the processor can execute on each packet prior to the arrival of the next packet.
- This severely limits the number of operations (and therefore the applications) that can be performed on the arriving packets.
- This example clearly brings out the inability of general-purpose processors in performing networking functions at line-rate.

# New Types of Network Applications

- Deep packet classification processing
  - Single field: a single header field is examined (e.g. packet forwarding)
  - Multi-field: multiple fields are examined (e.g. firewall, QoS applications like IntServ, Diffserv)
  - Deep: examines both headers and payload (server load balancing, intrusion detection, virus scanning)
  - Must be executed at *line rates*
- Security related applications
  - Web servers: e-commerce, banking, financial trading (SSL, TLS)
  - CPU intensive, 5 to 7 times more than other network applications
  - Must be executed at *line rates*

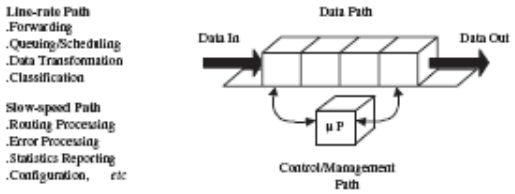
# Kernels of network applications

- Six categories [Yi2006]:
  - Pattern matching (e.g. lookup tables)
  - Lookup (e.g. IPv4 and IPv6 routing)
  - Computation (e.g. checksum)
  - Data manipulation (e.g. decrement TTL in IPv4 routing)
  - Queue management (e.g. packet dropping, shaping)
  - Control processing (e.g. table updates, statistics)*
- Data intensive and branch intensive
  - Ex: high load/store instruction ratio: Netbench TL, ROUTE, DRR, NAT (75%)
  - Ex: relatively high branch ratio: Netbench IPCHAINS (15.3%), CRC (2.5%)

(*) not at line rates

# Type of Network Application Tasks

- Type of tasks: data path, control path, management path
- Data path at line rate
  - Receiving transmitting from MAC devices
  - Packet forwarding, classification, queuing, scheduling
- Control path are less time-critical
  - Table maintenance, routing, signaling policy management
  - Typically present little data parallelism
- Management path
  - System initialization, configuration



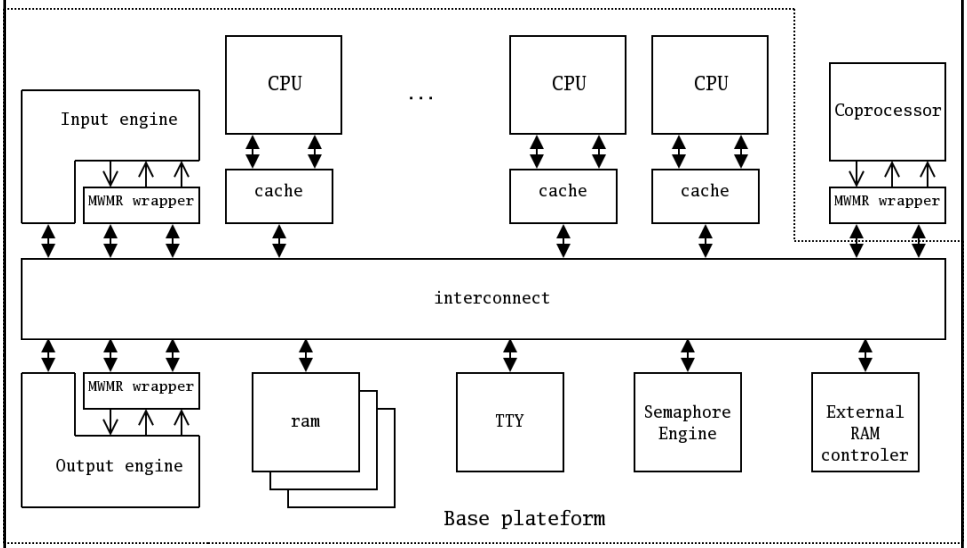

# Need of Scalability

- Performance scales well as the amount of processing increases
  - Increasing QoS requirements
  - Increasing number of media streams
- A scalable application architecture is obtained by distribution
  - Parallelizing application algorithms
  - Distribution of processing
- Distribution:
  - On-chip: exploit task level and packet level parallelism to distribute workload among processing elements
  - On-network: offload some task to network elements such as media gateway to improve computation-communication trade-off

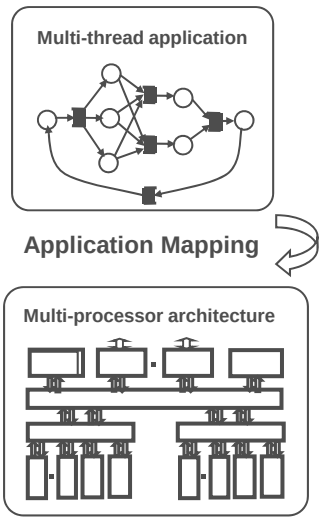
# Parallelism in Network Applications

- Net apps are typically layered (ISO-OSI model)
  - Produce parallelism
- Packet parallelism
  - *Packet-level parallelism*: each incoming packet is independent from others and can be processed concurrently (exploited in current commercial routers with IXP2800)
  - *Intra-packet parallelism*: different tasks processing a packet, for instance source and destination MAC address manipulation
- Packet dependency
  - It requires synchronization between packet processing
  - With TCP connections for encryption, state maintenance
  - For routing, address translation tables, traffic management counters
  - Example: for TCP flows, dependency probability of 14% in a window of 100 packets

# Generic Platform



# Mapping

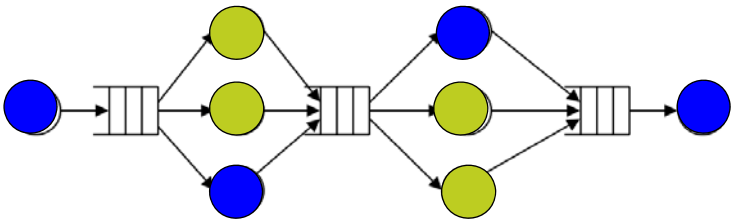



The system designer must have the following possibilities :

- choose the hard/soft implementation for each task
- map the software tasks on the programmable processors (and the hardware tasks on synthesized or existing coprocessors)
- map the communication channels onto the physical memory banks

# Application Model

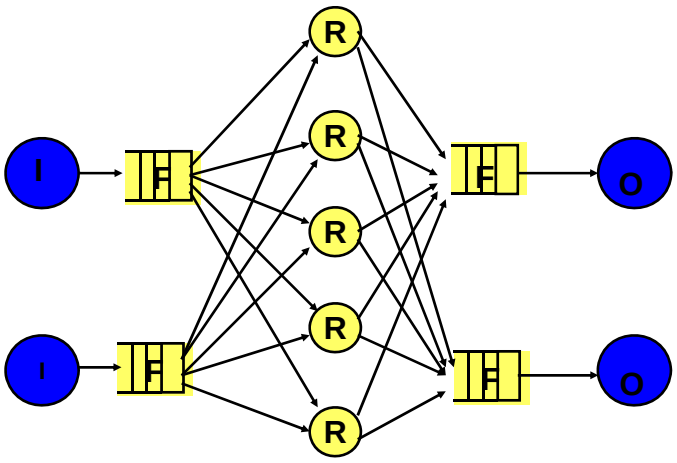
- The software parallel application is described as a task graph with two types of nodes : tasks & communication channels.
- Tasks communicate through Multi-Writer / Multi-Reader FIFOs..
- Tasks can be hardware or software.



# MWMR Communication Channels

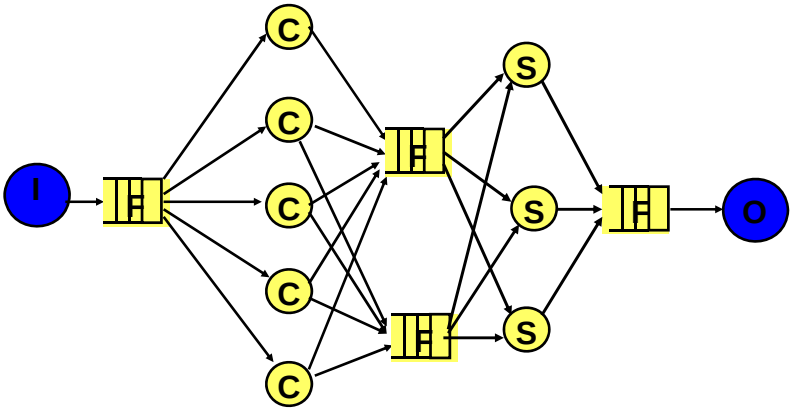
- Each MWMR channel is implemented as a software FIFO, and is caracterized by 2 parameters: width & a depth.
- Each MWMR channel is protected by a lock, in order to guarantee exclusive access.
- Read & Write communication primitives are non-blocking :
  - int mwmr_read(channel_id, *buffer, nb_bytes)
  - int mwmr_write(channel_id, *buffer, nb_bytes)
- As any task can be implemented in hardware or software, MWMR channels can be accessed by both hardware and software controllers.
- The software versions of the communication primitives are built upon the POSIX API : The software application can be executed on any UNIX workstation, before being mapped on the SoC.

# Task Structure Example




IPV4 Routing Application

# Task Structure Example




Classification Application

# Memory Management

- The network processor must have the largest possible storage capacity (several thousands packets).
- In networking applications, the relevant information is usually located within the first few bytes of a packet.
- On-chip memory is limited

→ External RAM is mandatory, with a careful allocation/free policy.

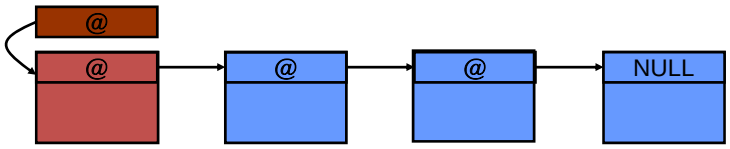
→ Only the packet descriptors are stored in the on-chip RAM.

# Memory Management

„Slot“ Data Structure

- Descriptor (128 bits)
- First slot (128 bytes)
- Following slots (128 bytes)

: MWMR channels
: on-chip RAM
: external RAM




# Use of Coprocessors

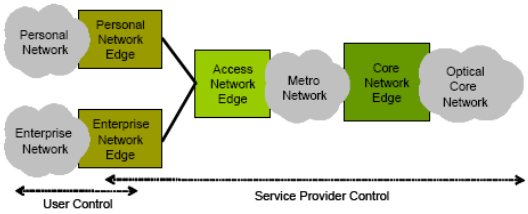
Both the Input Engine and Output Engine coprocessors are configured by software, and use a MWMR hardware controller.

- Input Engine
  - **Its aim is to copy the packets coming from the Gigabit Ethernet link into system memory.**
  - **It implements the management of the slot structure.**
- Output Engine
  - **Its aim is to reconstitute the packets from their slots in order to copy them to the outgoing Ethernet link.**
  - **The Output Engine works symmetrically to the Input Engine.**

# Examples

# Content Processing Networks (1/2)

- Content networking overcomes the inadequacies of existing networks by introducing intelligence into the network in order to enhance performance of services and delivery of content to the consumer
- Content Delivery Networks (CDN) are typically implemented as overlay networks and contain one or more nodes that can inspect and/or manipulate information in higher networking layers (four through seven in the OSI reference model).
- Examples of CDNs include web cache networks and video streaming networks. Examples of content networking devices include voice packet gateways, web caches, load-balancing switches and firewalls

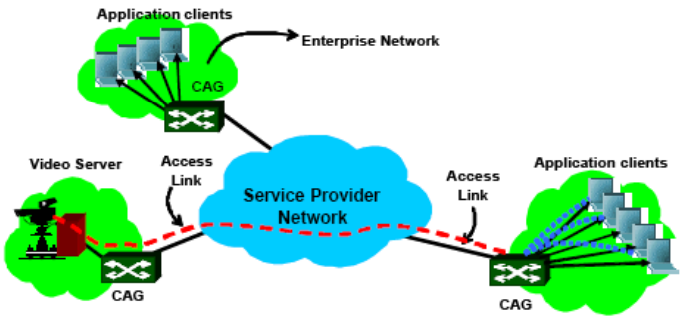


# Content Processing Networks (2/2)

- The network edge is typically characterized by an “impedance mismatch”. The mismatch is typically due to differences in physical network attributes such as network size, link bandwidth, network latency, network capacity or abstract attributes such as trust, authority.
- Such discrepancies in the network boundaries can be overcome by additional processing of the traffic flowing across them. However, this additional processing is often not the same for all traffic types, and is typically not known a priori. Content Networking applications adapt the processing to the type of traffic flowing across a given network edge.

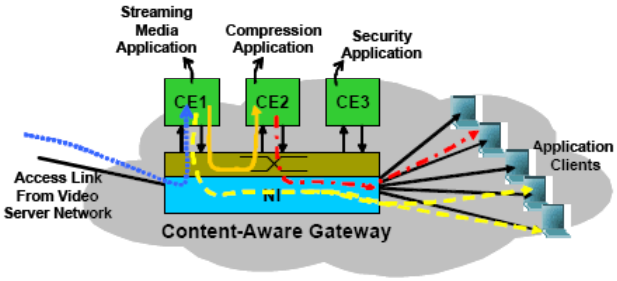
# CAG

- Streaming media delivery over context aware gateway (CAG)
  - The SMD application on the source CAG edge node replicates and unicasts the source video stream to the destination CAG edge device at each of the client sites. Thus, the application does not have to perform any multicasting and the bottleneck problem at the source access link is avoided.



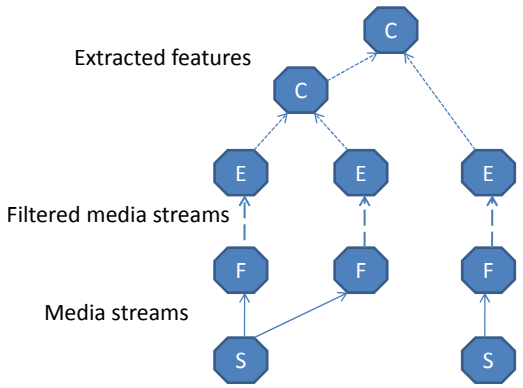
# Mapping Tasks on CAG

- Streaming media application
  - Delivers contents to clients
- Compression application
  - For clients requesting compressed media streams
- Encryption can be cascaded as well




# RTVCA

- Real time video content analysis
  - Surveillance systems, traffic control

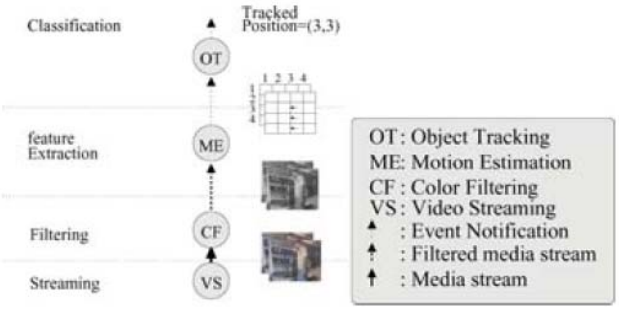



C: Classification
E: feature Extraction
F: Filtering
S: Streaming

Directed Acyclic Graph (DAG)
(tasks + flows of data)

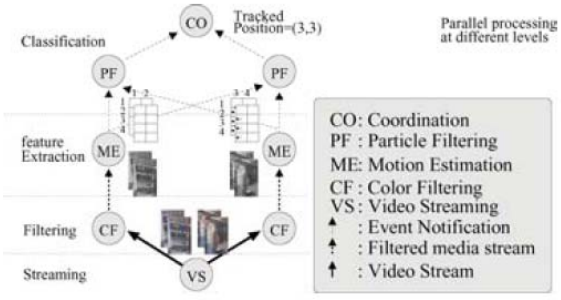
# Parallelization Example 1

- Real time tracking of moving object in a video stream
- It can be executed in parallel by four processors




# Parallelization Example 2

- Finer granularity
- Focusing on processing bottlenecks
  - Motion estimation: 2 processors
  - Classification: 3 processors



# INTRO II

Software Challenges

# Reliability and Partitioning

- Shift to multicore in embedded systems
  - Stability, reliability, performance assurance, low-power
  - Rich and heterogeneous application scenario
  - GP OSes + real-time OSes
- Reliability
  - Performance assurance = reliability in normal operations
  - Robustness = reliability in abnormal situations
- Two approaches
  - Hardware (multicore)
  - Software (OS scheduling and system protection)

40

# Reliability

- Reliability of embedded systems can be enhanced by multicore and partitioning approaches
  - Physical partitioning by multicore AMP with distributed memory
  - Logical partitioning by processor virtualization and SMP
- More flexible and scalable platforms

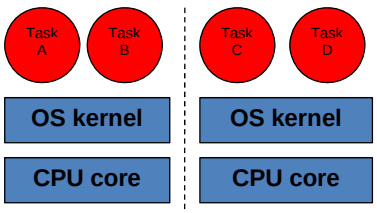
41

# Partitioning

- Multitasking parallel model
  - CPU core level partitioning
  - OS kernel level partitioning
- An independent instance of the OS kernel runs on each CPU core in an AMP type of multicore system
  - Isolation between interrupt driven real time and CPU centric application
- On SMP, hardware facilities allow

  to run multiple OSes

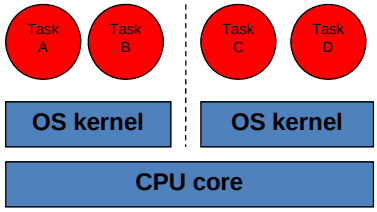
  [Richard Low, Sept. 2005]




42

# Logical Partitioning

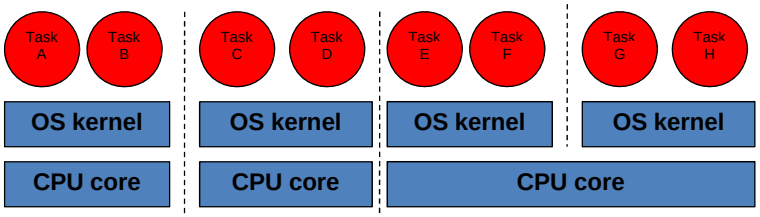
- Partitioning by physical cores has limitations
  - Flexibility is an issue (application dependency)
- Go to logical partitioning
  - Virtualization (multiple Oses on the same CPU core)
- AMP is not necessarily the best architecture, SMP may be preferable, SMP can be more scalable
  - Go to SMP architectures
  - Pure SMP OS are not efficient




43

# Hybrid SMP/AMP

- SMP OS applies AMP scheduling
- Specific tasks are bounded to defined CPU using CPU-affinity functions
- Heterogeneous OS environment
- Balance reliability and flexibility
  - Proper control of scheduler
  - Assignment of resources



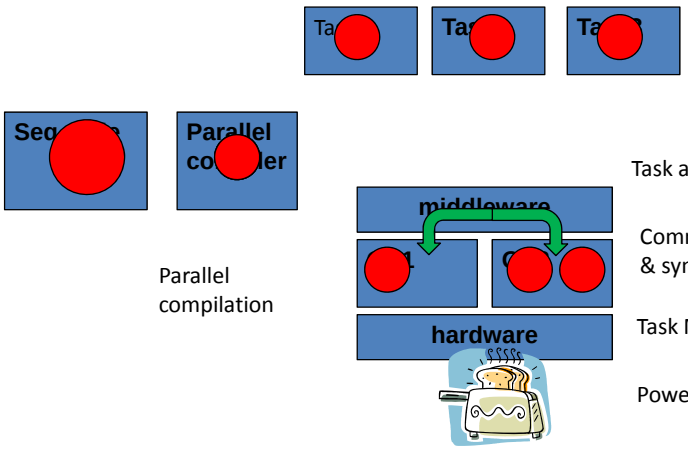

44

# Software Design Challenges (1/2)

- Compiler enabling automatic task level parallelism in the Multitasking parallel world
- Partitioning vs Communication
  - Need OS wrapping for communication between different execution domains
  - Using compatible API with local execution domain (such as SysV IPC)
- Task allocation and migration locally and between execution domains
- Resource assignment architectures with multiple voltage and frequency domains (GALS)

45

# Software Design Challenges (2/2)




46