Lecture 6: Hardware

Michael O’Boyle
Embedded Software

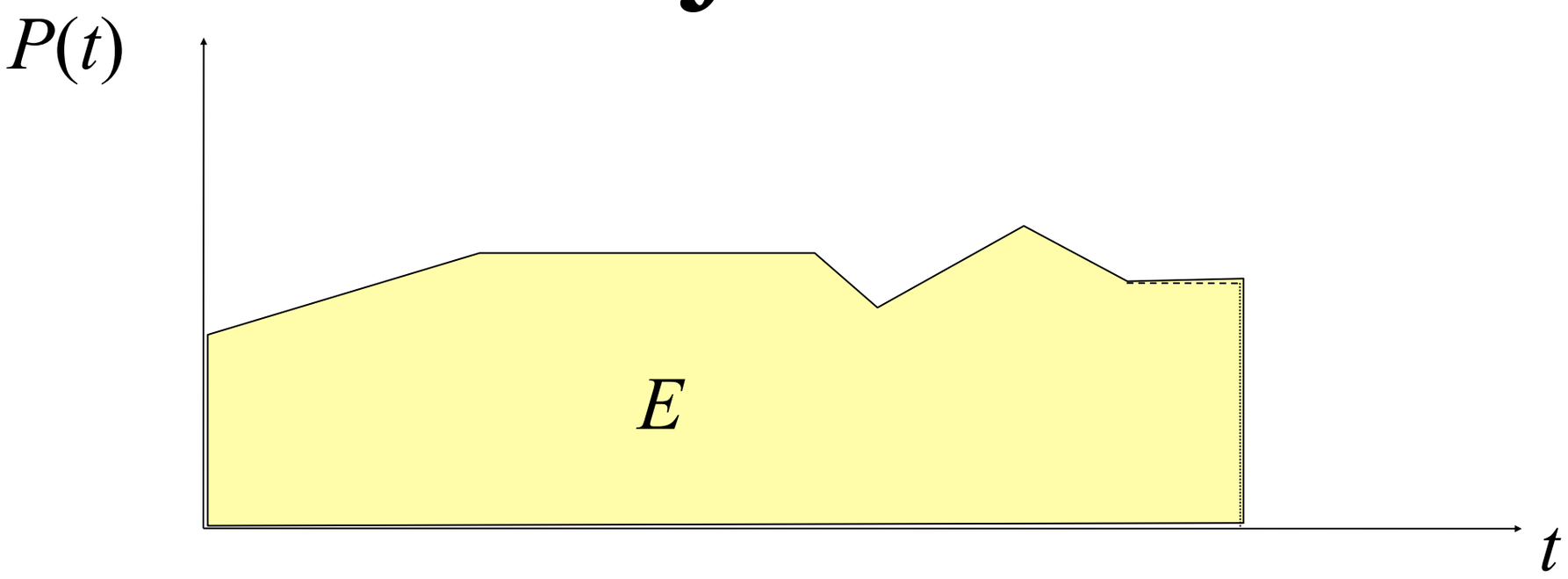
Overview

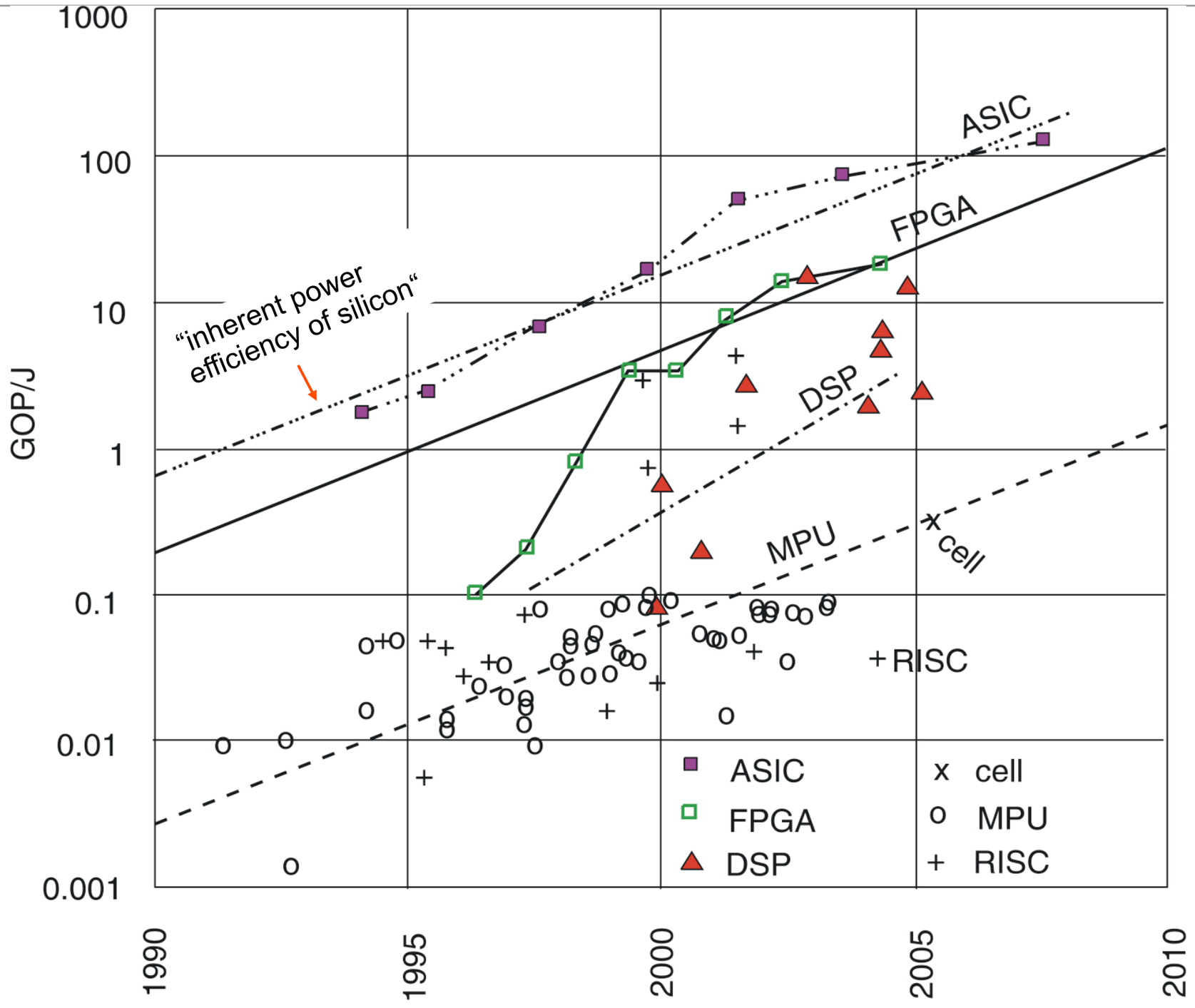
- Power and Energy
  - Processors
    - Energy Efficiency
    - Code Size
  - DSPs
    - Address generating Units
    - Specialised Arithmetic
  - Multimedia Instructions
    - VLIW
  - Reconfigurable

Power and Energy

- Energy and Power are first class issue in embedded and systems design
  - Battery life
  - Energy density, thermal
  - Environment - no more data centres in London
- Energy not always the same
  - A processor that is more power-hungry but takes less time may use less energy

$$E = \int P(t)\, dt$$



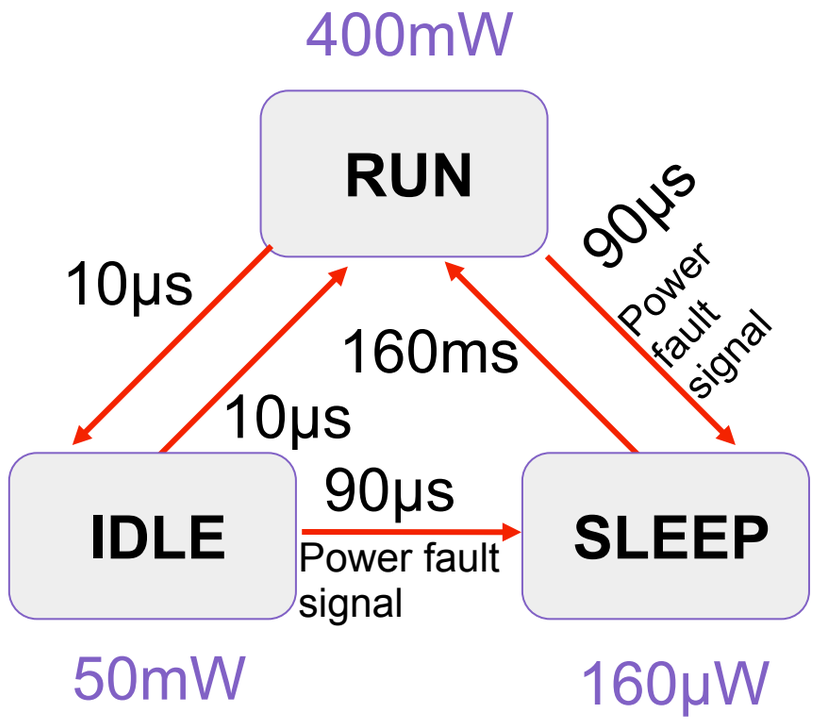

1000
100
10
1
0.1
0.01
0.001
GOP/J
1990
1995
2000
2005
2010
ASIC
FPGA
DSP
MPU
cell
RISC
"inherent power efficiency of silicon"
ASIC
FPGA
DSP
x cell
o MPU
+ RISC

Processors

- Vast majority of embedded systems based on (semi-)programmable processors rather than specialised hardware (ASICs)
  - Ease of programming, upgrade or change of use
- Variety of ways to manage power

Example: STRONGARM SA1100

RUN: operational
IDLE: a SW routine may stop the CPU when not in use, while monitoring interrupts
SLEEP: Shutdown of on-chip activity



Dynamic Voltage Scaling

Power consumption of CMOS circuits (ignoring leakage):

$P = \alpha\ C_L V_{dd}^2\ f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS

$$\tau = k C_L \frac{V_{dd}}{(V_{dd} - V_t)^2}$$ with

V_t : threshhold voltage

(V_t < than V_{dd})

- Decreasing voltage slows down linearly, quadratic power saving
- Intel SpeedStep has 6 speed/voltage settings
- ARM has big.LITTLE offering - c 18 power levels!

Multi-core: Reduce Power for same performance?

Rough approximations

Basic equations

Power: $P \sim V_{DD}^2$,

Maximum clock frequency: $f \sim V_{DD}$,

Energy to run a program: $E = P \times t$, with: t = runtime (fixed)

Time to run a program: $t \sim 1/f$

Changes due to parallel processing, with β operations per clock:

Clock frequency reduced to: $f' = f / \beta$,

Voltage can be reduced to: $V_{DD}' = V_{DD} / \beta$,

Power for parallel processing: $P^\circ = P / \beta^2$ per operation,

Power for β operations per clock: $P' = \beta \times P^\circ = P / \beta$,

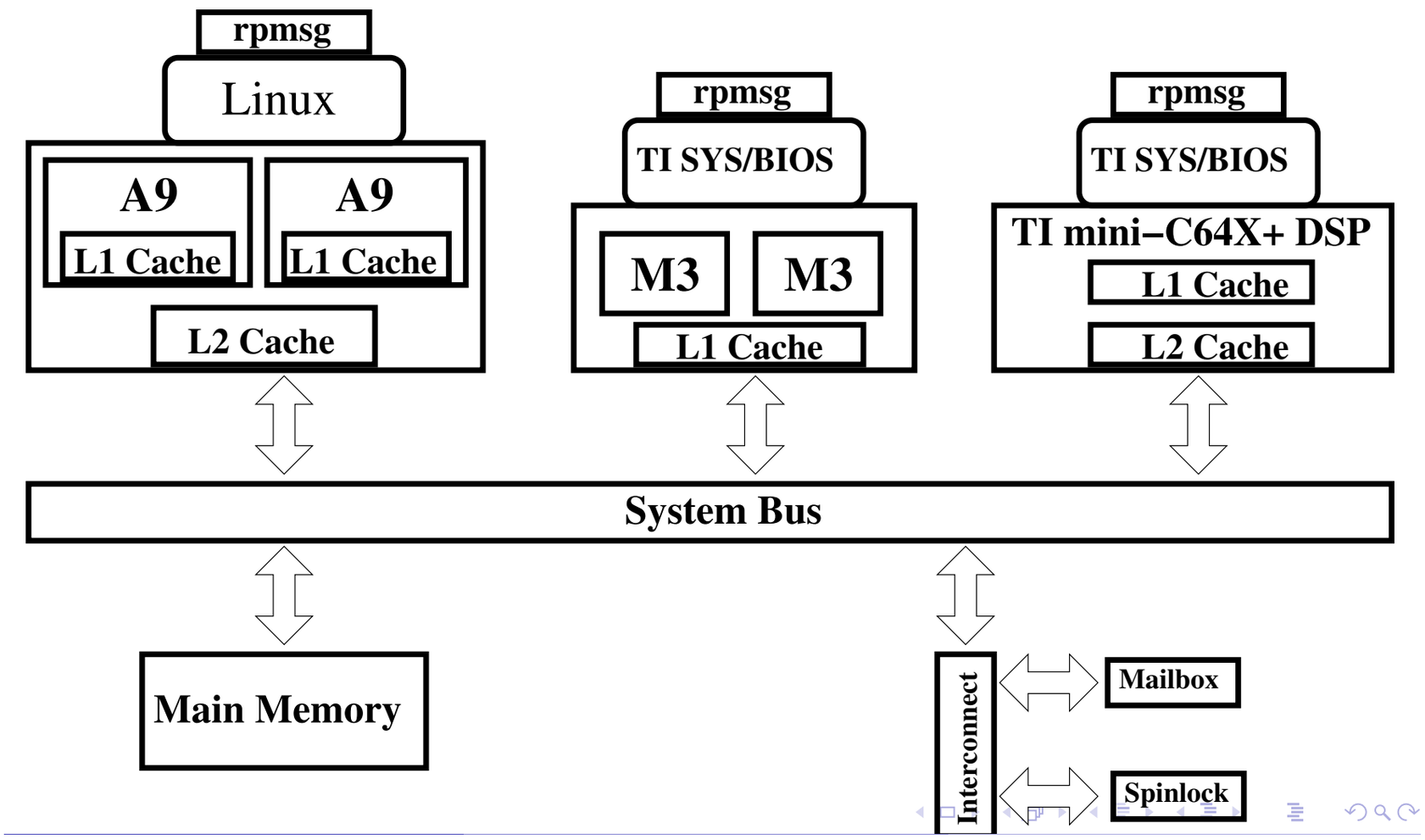
Time to run a program is still: $t' = t$,

Energy required to run program: $E' = P' \times t = E / \beta$

Argument in favour of voltage scaling and parallel processing

Detour in to real power experiments on heterogeneous multicores

Heterogeneous system

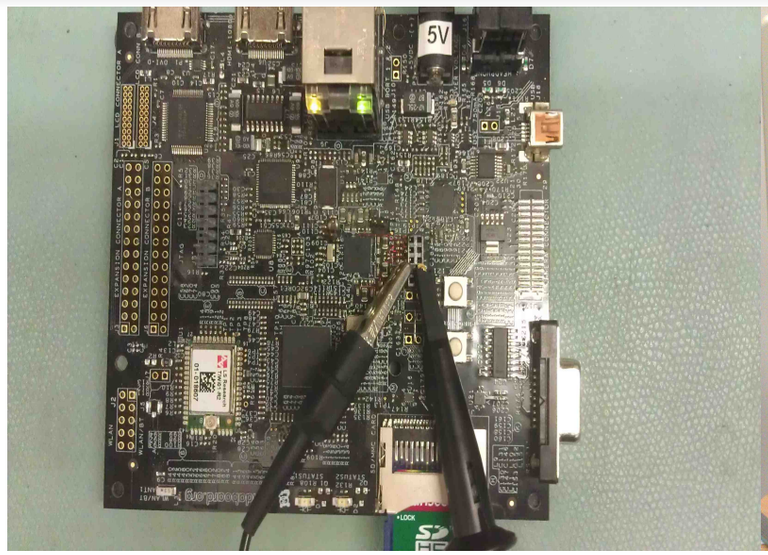


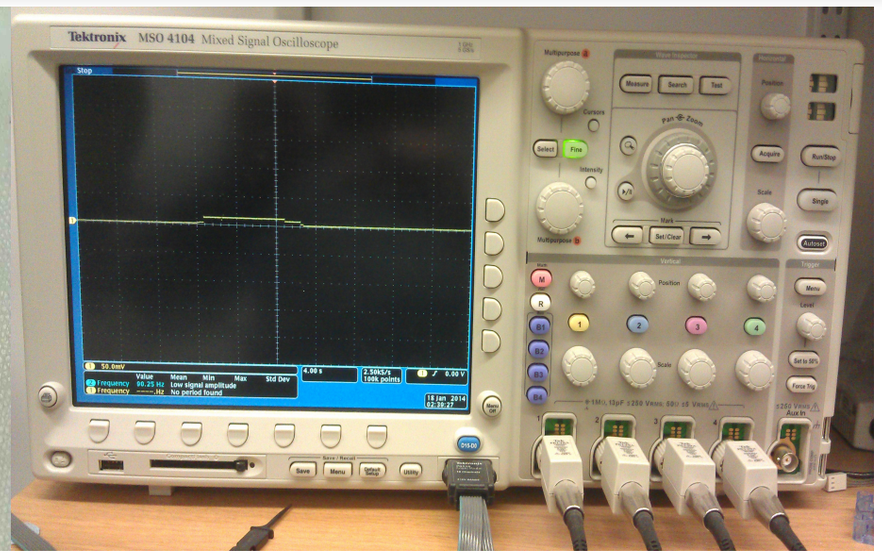
OMAP4

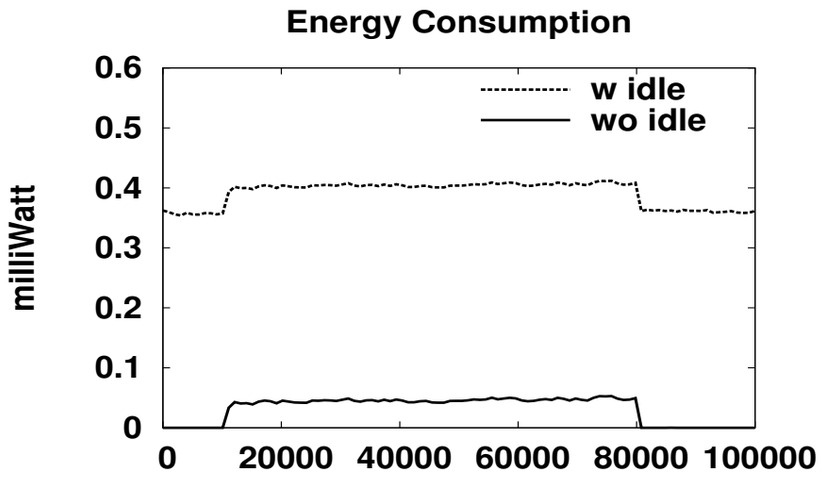
Want to partition programs to minimize time and energy

Challenging hardware, best partition depends on criteria

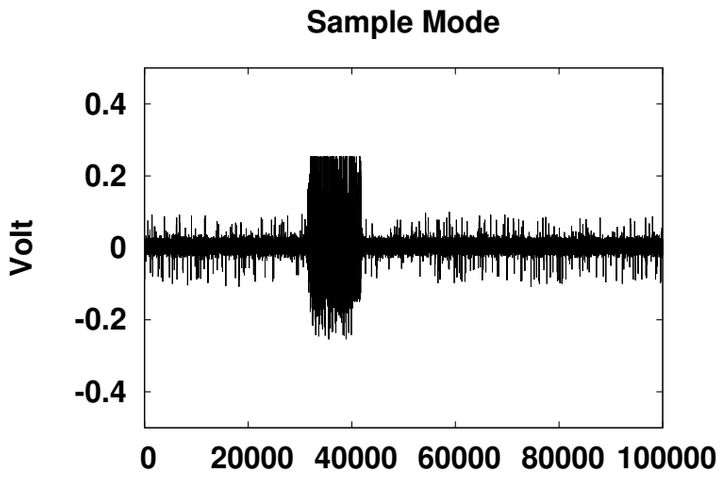
Measuring energy



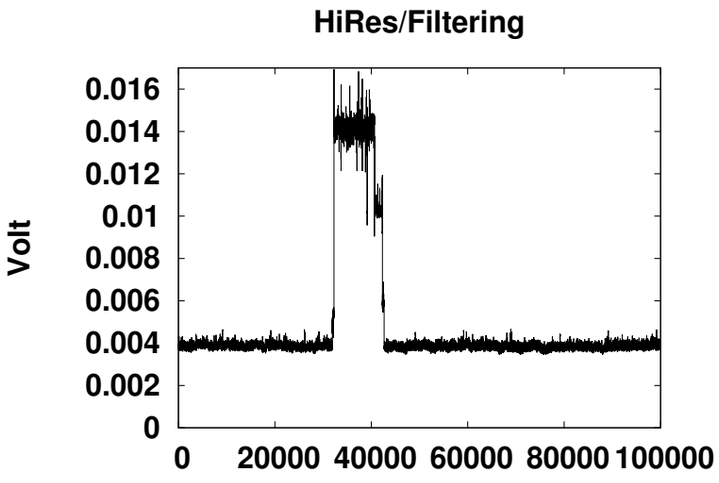




(a) with vs without idle

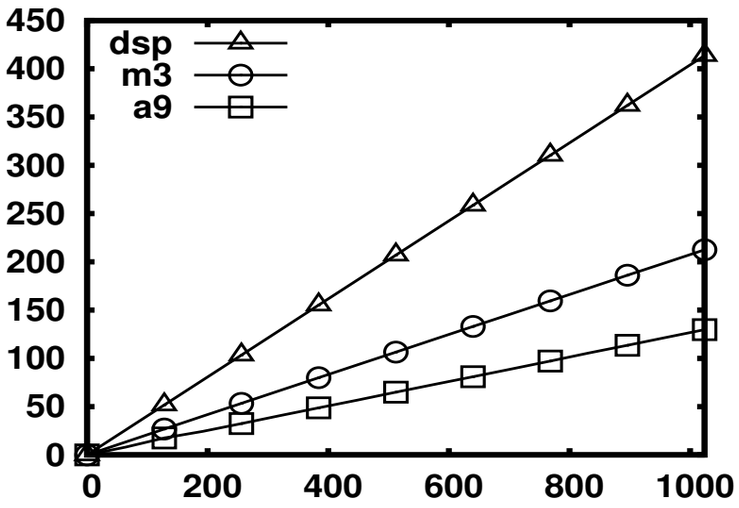


(b) sample

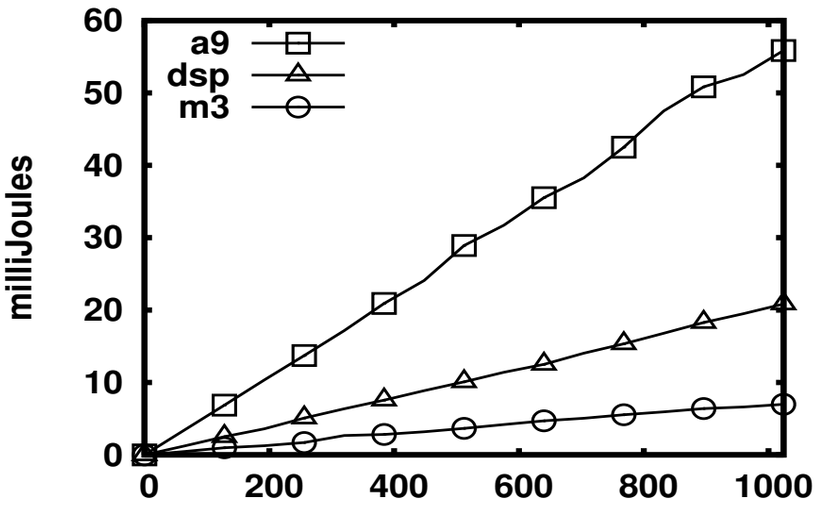


(c) filtering

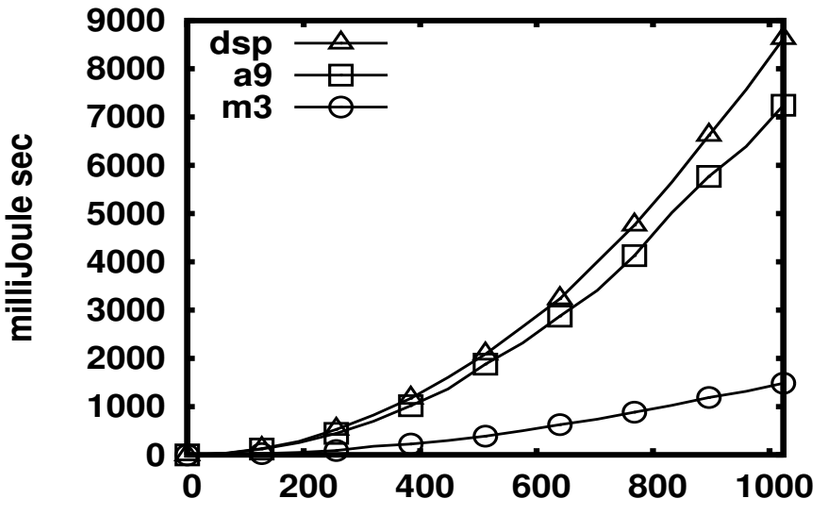
MxM on a single core, no idle



(a) Runtime



(b) Energy wo.static



(c) EDP wo.static

Unoptimised

Runtime : A9>M3>DSP

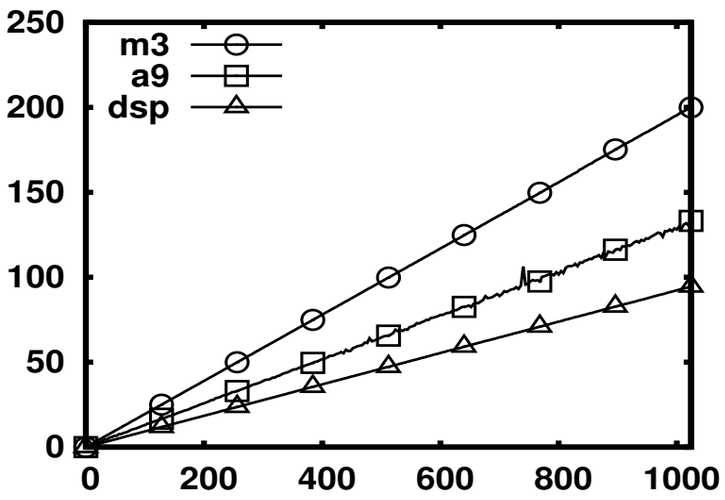
Energy: M3>DSP>A9

ED: M3>>A9>DSP

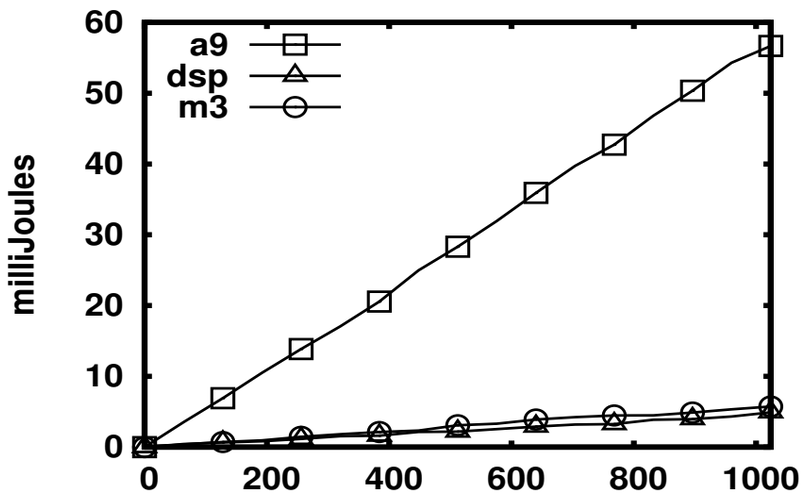
Conclusion

Use A9 or M3, turn off DSP

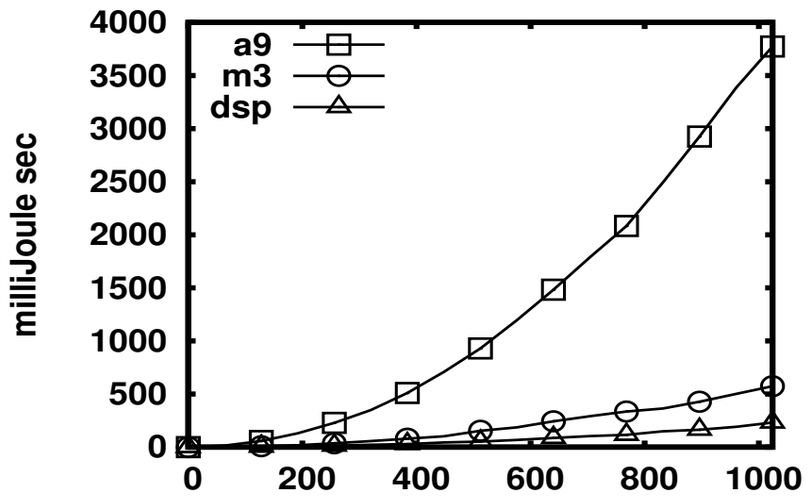
MxM: optimized, no idle power



(a) Runtime Opti



(b) Energy wo.static



(c) EDP wo.static

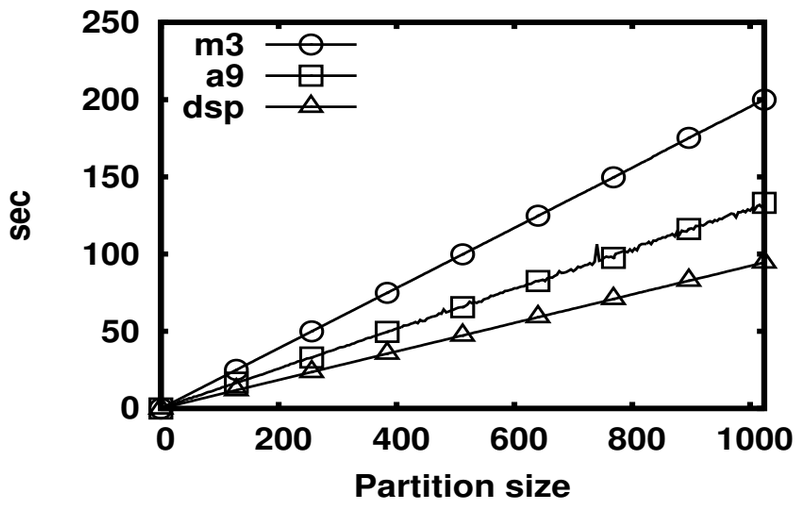
Optimised Runtime : DSP>A9>M3

Energy : DSP/M3>A9

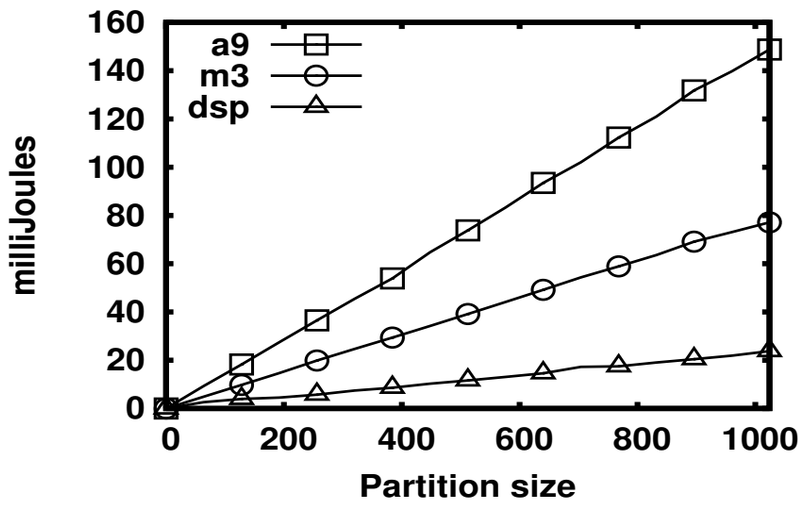
ED : DSP>M3>>A9

Conclusion DO NOT Use A9 use DSP

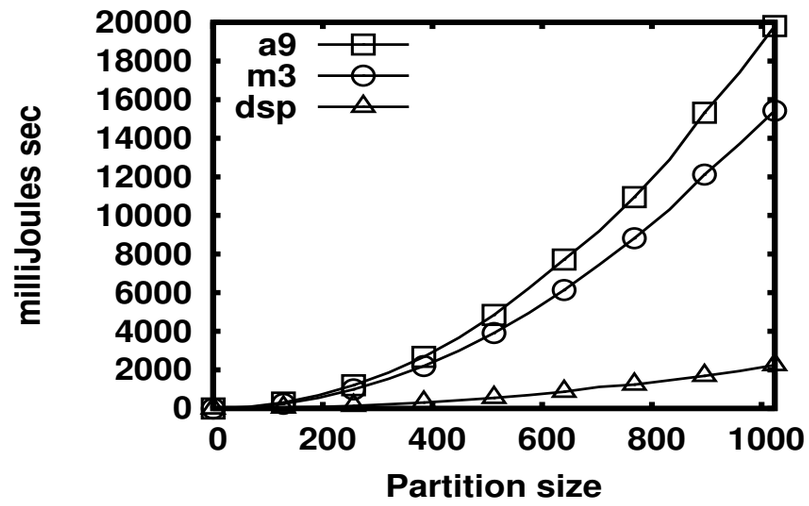
MxM: including idle power



(a) Runtime Opti



(b) Energy w.static



(c) EDP .static

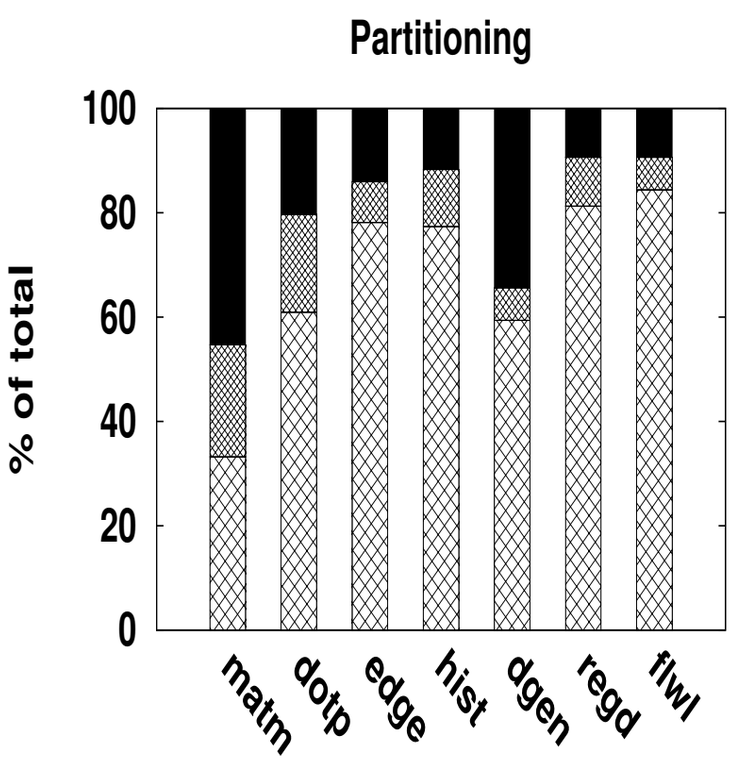
Optimised Runtime : DSP>A9>M3

Energy : DSP >M3>A9

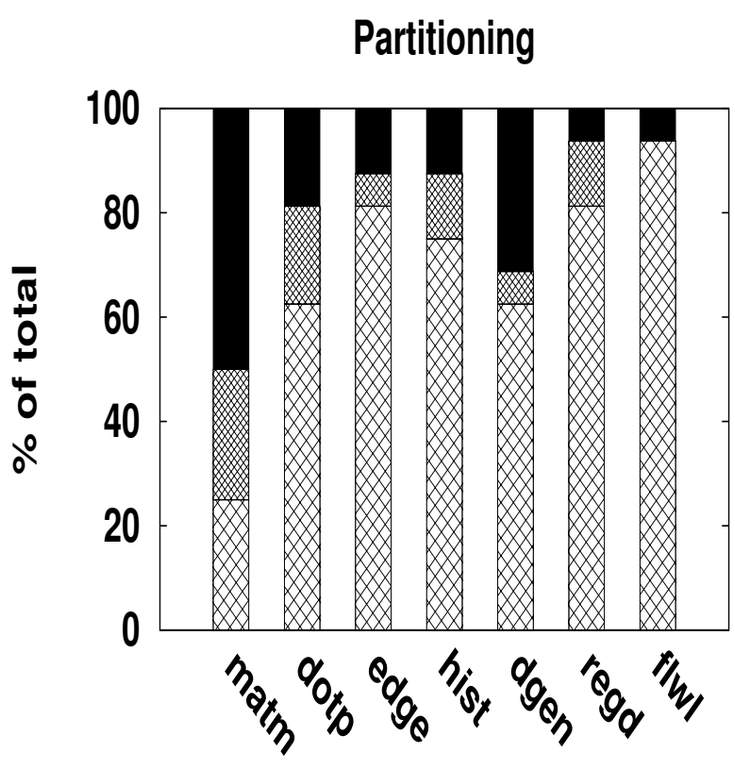
ED : DSP>>M3/A9

Conclusion DO NOT Use M3/A9 use DSP

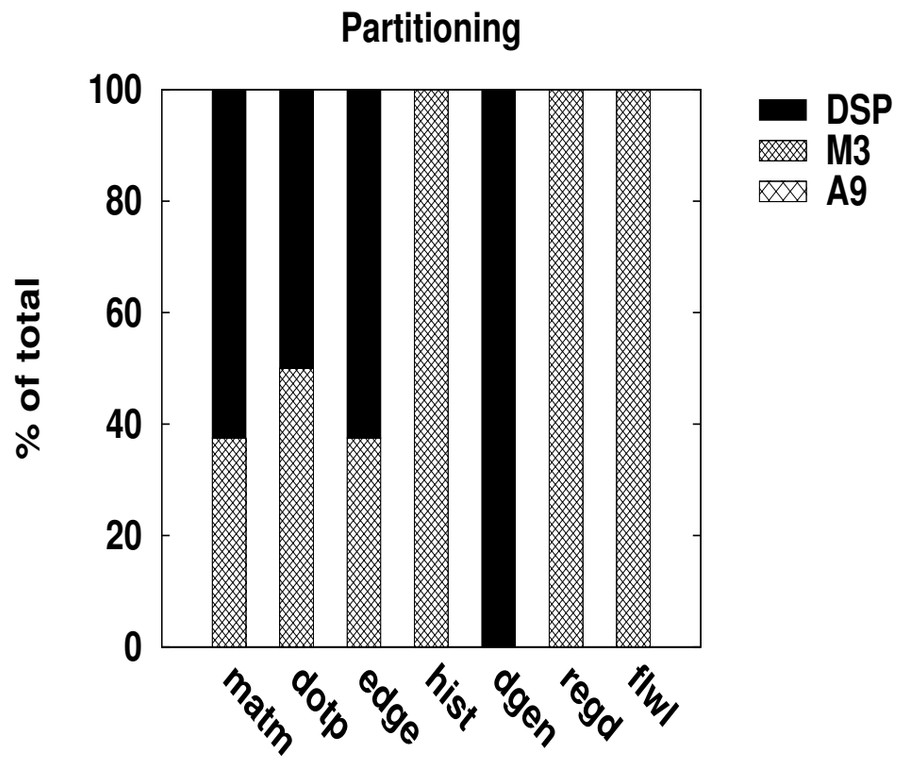
Best Partition across programs



(a) time



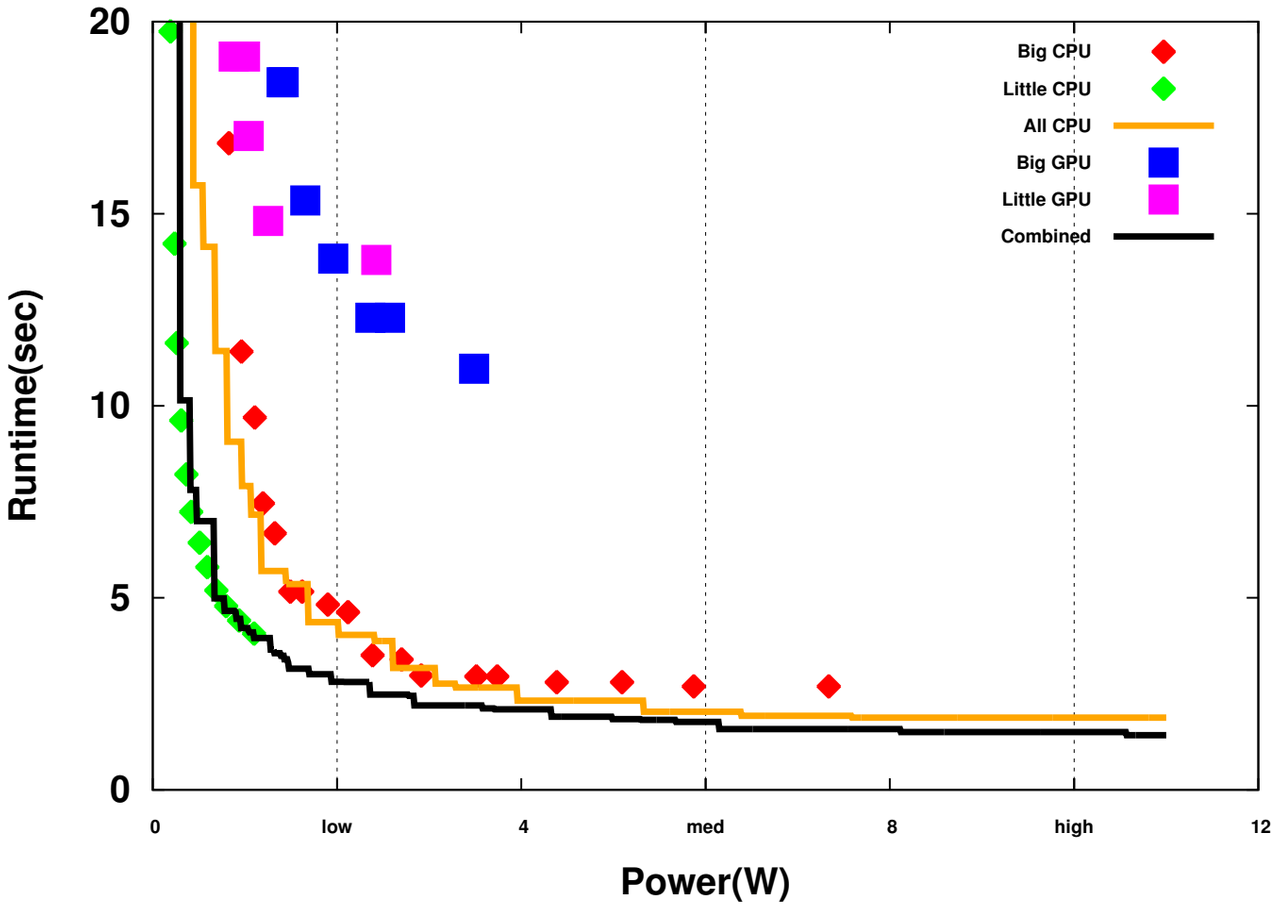
(b) with idle



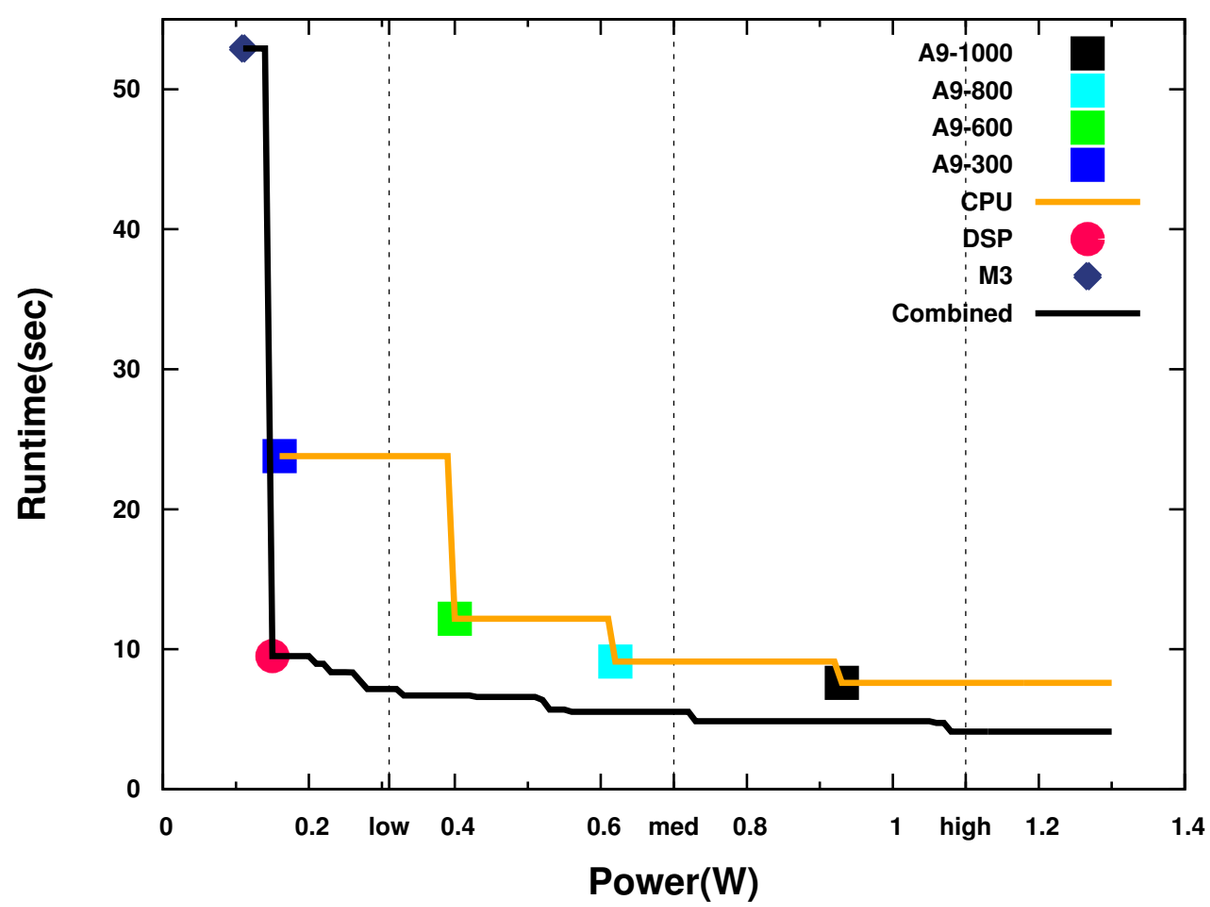
(c) wo idle

Best partition/cores varies with programs/optimization

Removal of idle by gating has big impact



(a) Exynos



(b) OMAP4

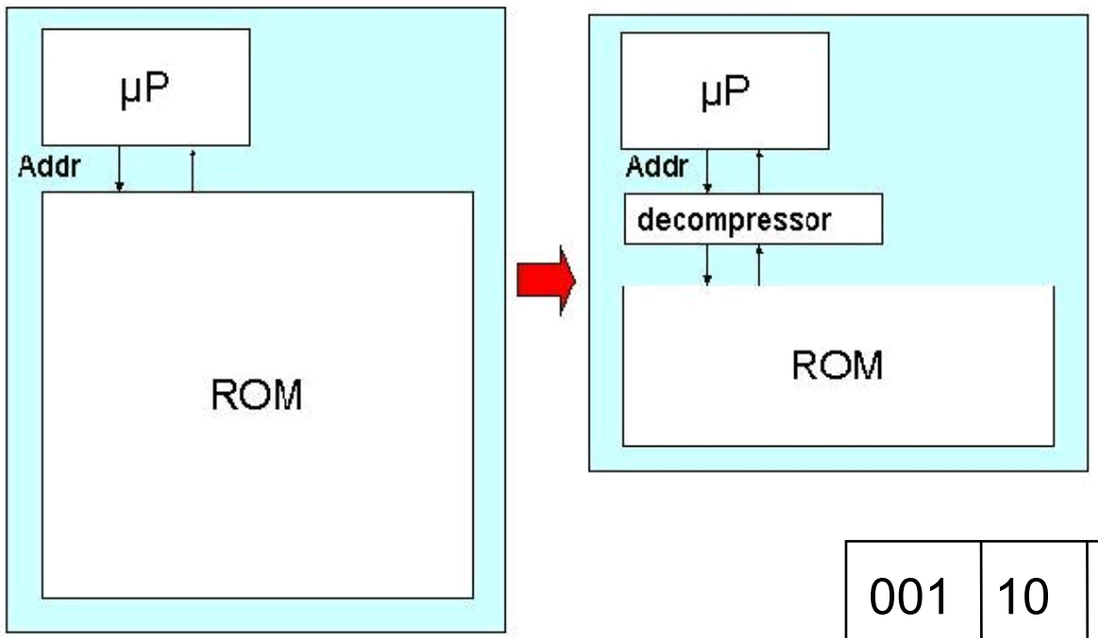
Figure : Power/Performance pareto : Using all the processors gives better pareto points than using the CPU only with *DVFS*.

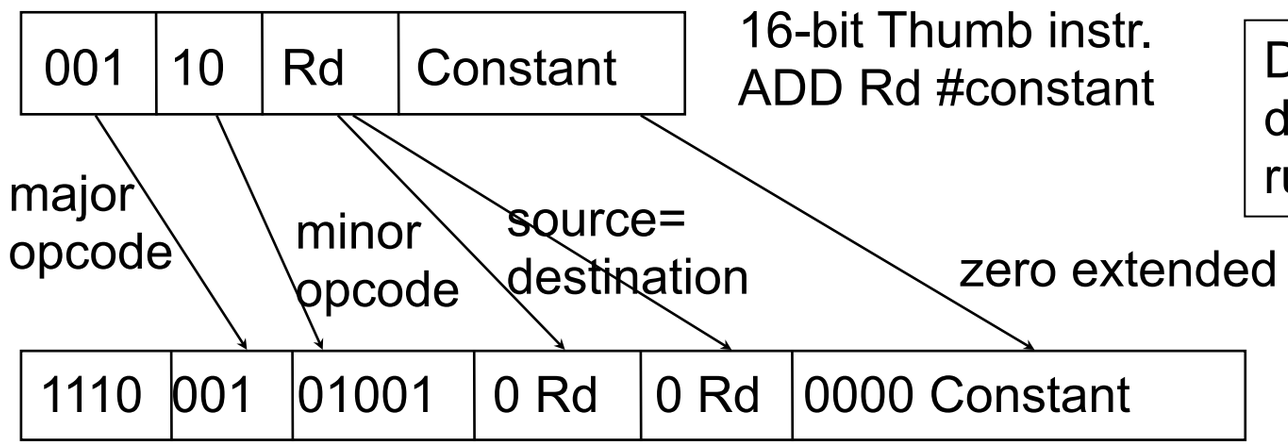
Multicores can save power. Minimising energy/power hard

End of detour

Code Size

- Memory costs. Reducing code size in ROM can be significant. CISC helps





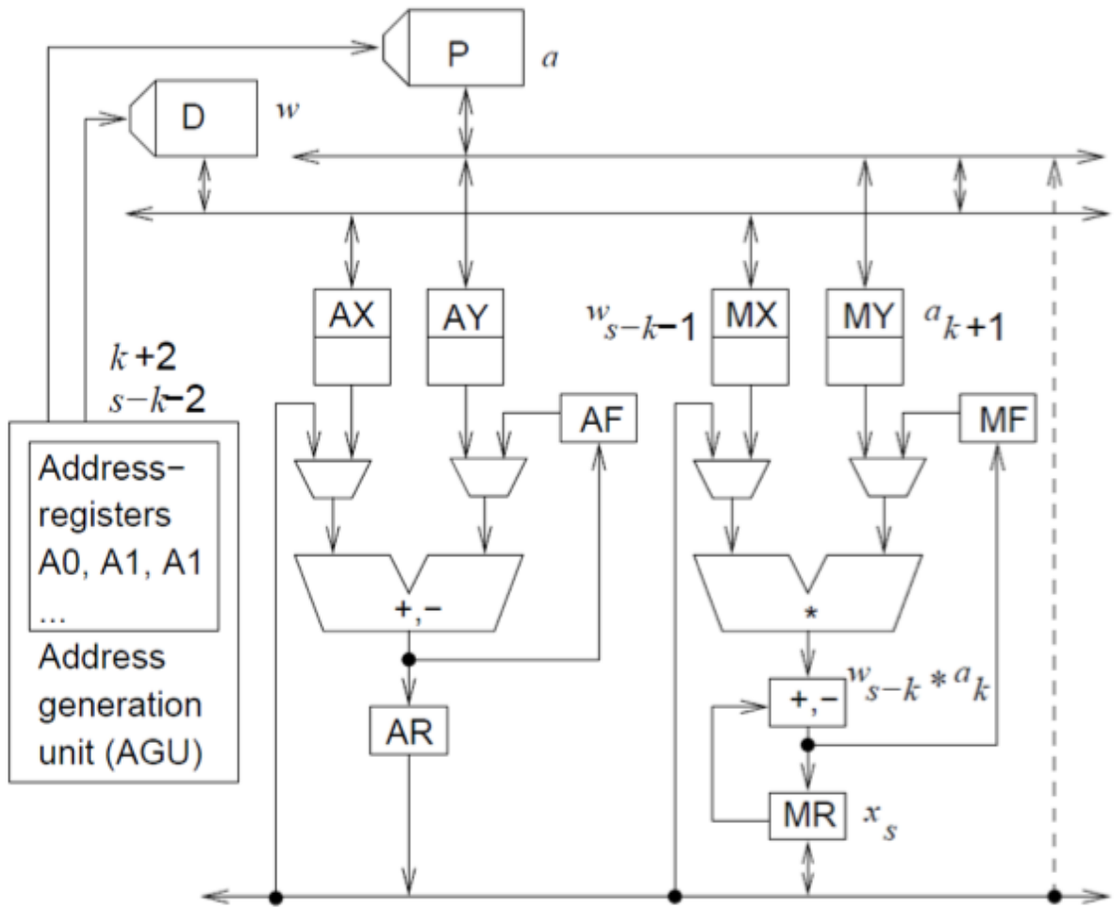
- Reduction to 65-70 % of original code size
- 130% of ARM performance with 8/16 bit memory
- 85% of ARM performance with 32-bit memory

Specialised Processors

- Filtering in digital signal processing. Signal at t(s) depends weighted avg of k previous inputs

$$x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k$$

ADSP 2100



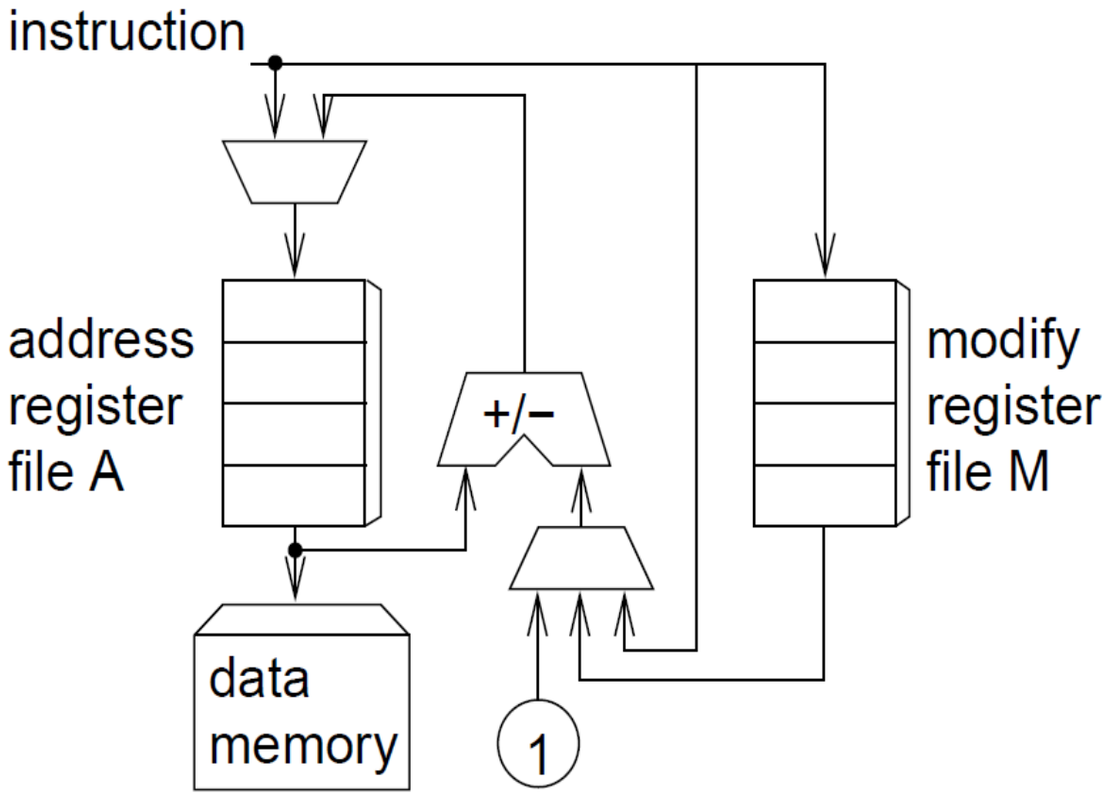

```
-- outer loop over
-- sampling times t_s
{ MR:=0; A1:=1; A2:=s-1;
  MX:=w[s]; MY:=a[0];
  for (k=0; k <= (n−1); k++)
   { MR:=MR + MX * MY;
     MX:=w[A2]; MY:=a[A1];
     A1++; A2--;
    }
 x[s]:=MR;
}
```

Maps nicely but specialised

DSPs

- Specialised processors found in many embedded settings
- Have a number of special features
  - Specialised addressing modes
  - Separate addressing Unit
  - Saturating Arithmetic
  - Fixed Point Arithmetic
  - Real-time capabilities
  - Zero-overhead loops
  - Multiple memory banks
  - Heterogeneous register files
  - Multiply/accumulate instructions

Address generating Units



- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- A := A ± 1 also takes 0 time,
- same for A := A ± M;
- A := <immediate instruction> requires extra instruction

Minimize load immediates

Specialised Arithmetic

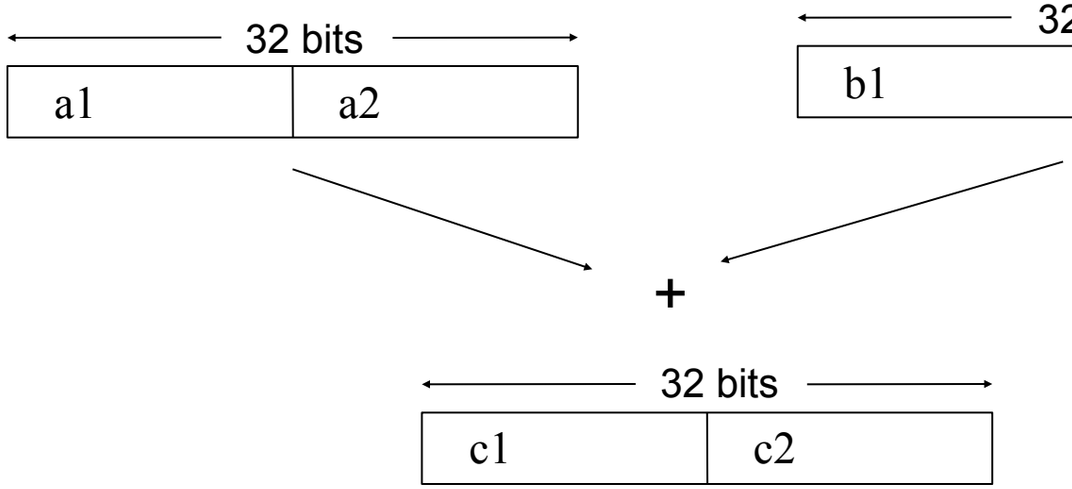
- Returns largest/smallest number in case of over/ underflows
- Example:

a		0111
b	+	1001
standard wrap around arithmetic	(1)	0000
saturating arithmetic		1111
(a+b)/2: correct		1000
wrap around arithmetic		0000
saturating arithmetic + shifted		0111

- Appropriate for DSP/multimedia applications: "almost correct"
  - No timeliness of results if interrupts are generated for overflows
  - Precise values less important
  - Wrap around arithmetic would be worse.

Multimedia Instructions

- Multimedia instructions exploit many registers, adders etc are quite wide (32/64 bit), whereas most multimedia data types are narrow
- 2-8 values can be stored per register and added. E.g.:

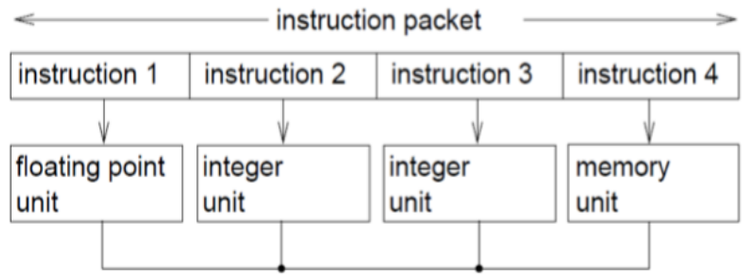


2 additions per instruction;
no carry at bit 16

- Cheap way of using parallelism
- SSE instruction set extensions, SIMD instructions

VLIW

- Instructions included in long instruction packets.
- Instruction packets are assumed to be executed in parallel.
- Fixed association of packet bits with functional units.



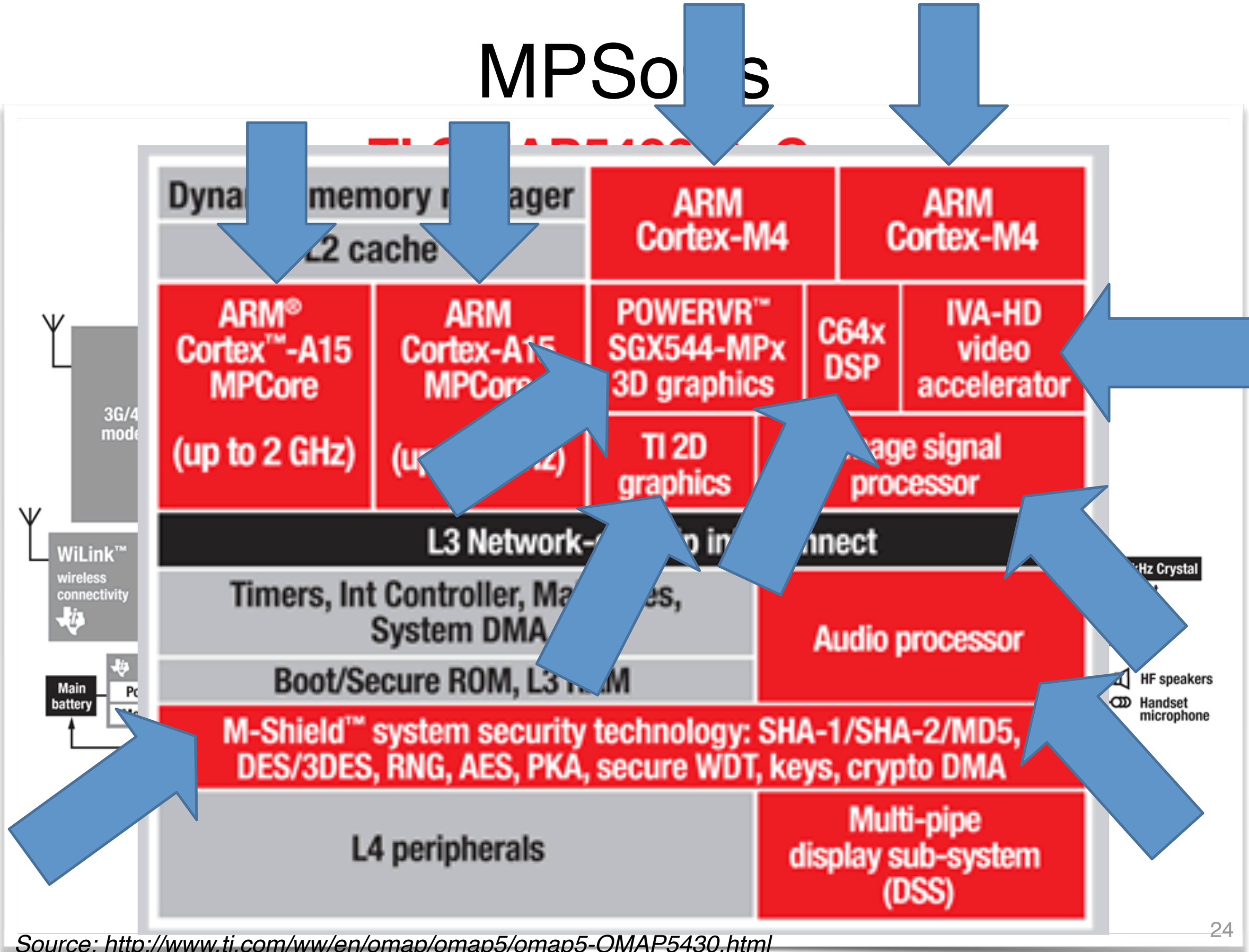
- Compiler is assumed to generate these “parallel” packets
- Complexity of finding parallelism is moved from the hardware (RISC/CISC processors) to the compiler;
- Ideally, this avoids the overhead (silicon, energy, ..) of identifying parallelism at run-time.

A lot of expectations into VLIW machines

- However, possibly low code efficiency, due to many NOPs

Explicitly parallel instruction set computers (EPICs) are an extension of VLIW architectures: parallelism detected by compiler, but no need to encode parallelism in 1 word.

MPSoCs



Source: http://www.ti.com/ww/en/omap/omap5/omap5-OMAP5430.html

Reconfigurable

- Custom HW may be too expensive, SW too slow.

Combine the speed of HW with the flexibility of SW

- HW with programmable functions and interconnect.
- Use of configurable hardware;
  common form: field programmable gate arrays (FPGAs)

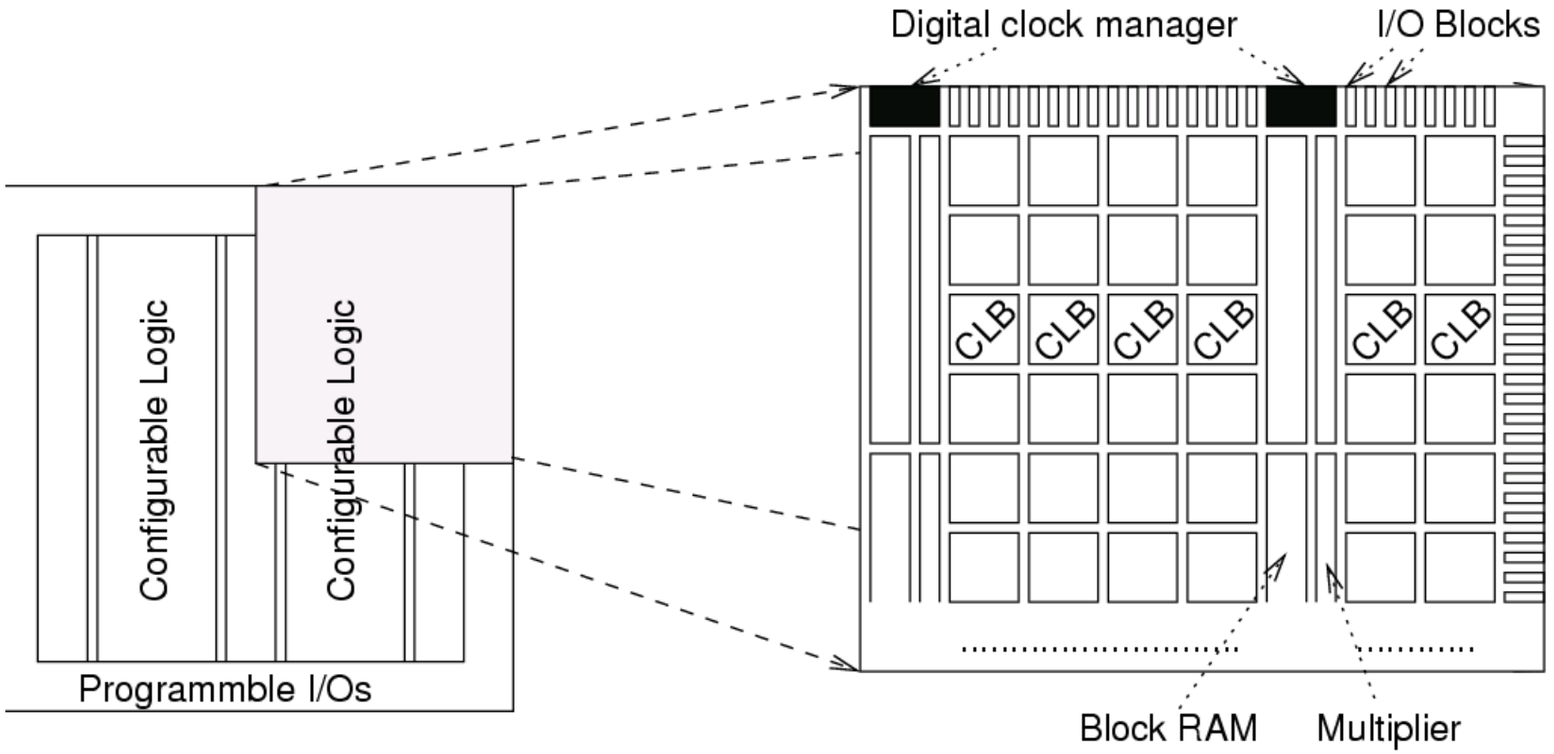
Applications:

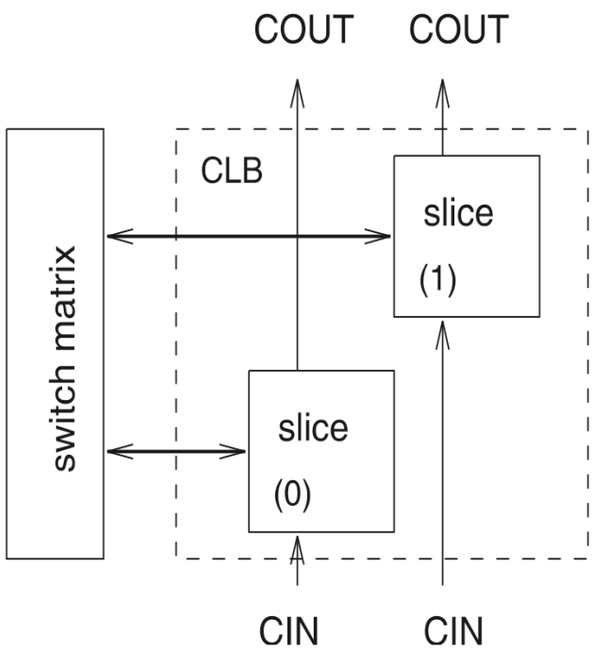
- algorithms like de/encryption,
- pattern matching in bioinformatics,
- high speed event filtering (high energy physics),
- high speed special purpose hardware.

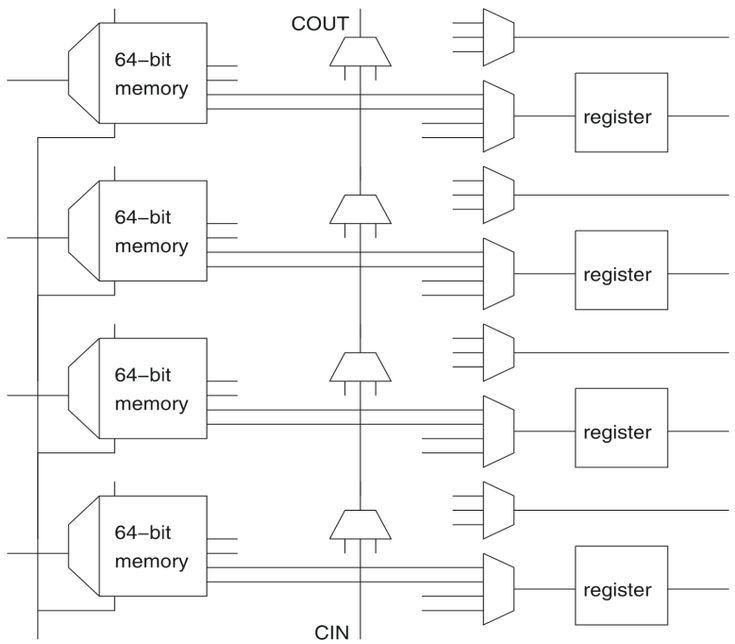
Very popular devices from

- XILINX, Actel, Altera and others

Reconfigurable Floor plan







Memories typically used as look-up tables to implement any Boolean function of ≤ 6 variables.

Processors typically implemented as “soft cores” (microblaze)

Summary

- Power and Energy
  - Processors
    - Energy Efficiency
      - Heterogeneous multicores
      - Code Size
- DSPs
  - Address generating Units
  - Specialised Arithmetic
- Multimedia Instructions
  - VLIW
- Reconfigurable