Energy-Aware Computing

Lecture 9: Dynamic power management and DVFS

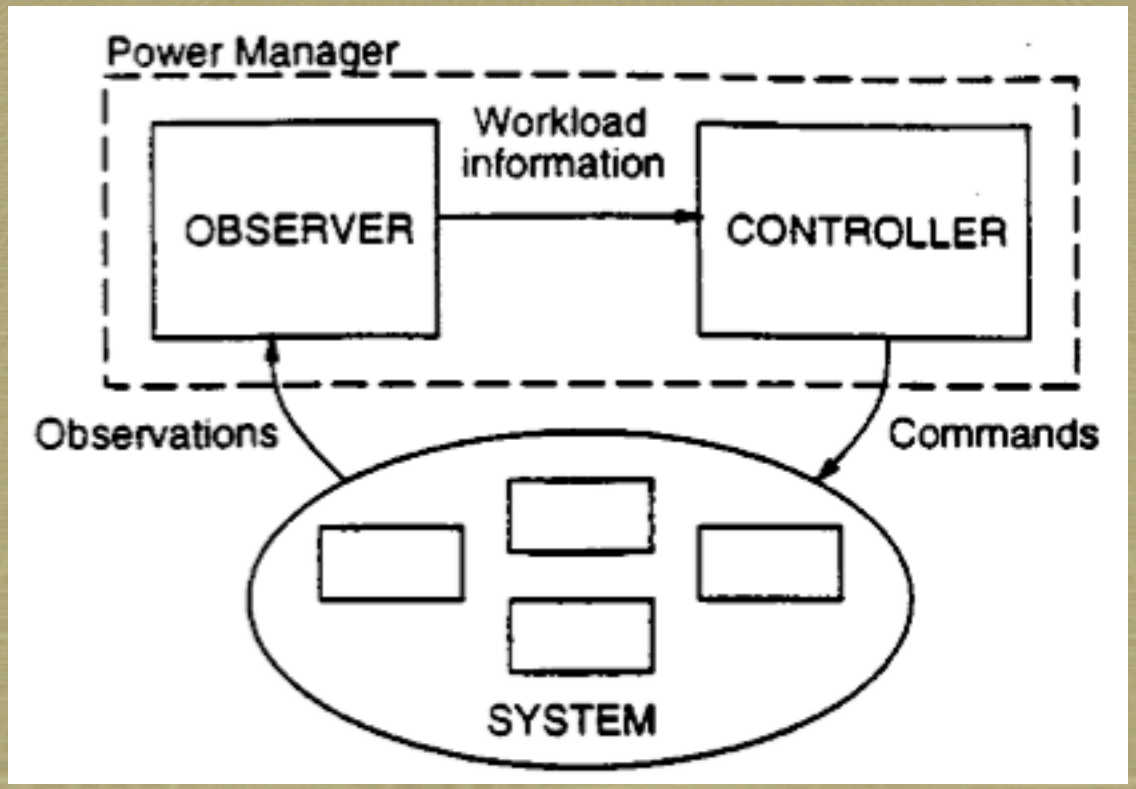
Outline

- Dynamic power management (DPM)
  - Power states and transitions
  - Break-even time
  - Time out policy
- Dynamic Voltage-Frequency Scaling
  - Voltage scheduler
- Multiple-clock domain processors
  - With local DVS

Dynamic Power Management

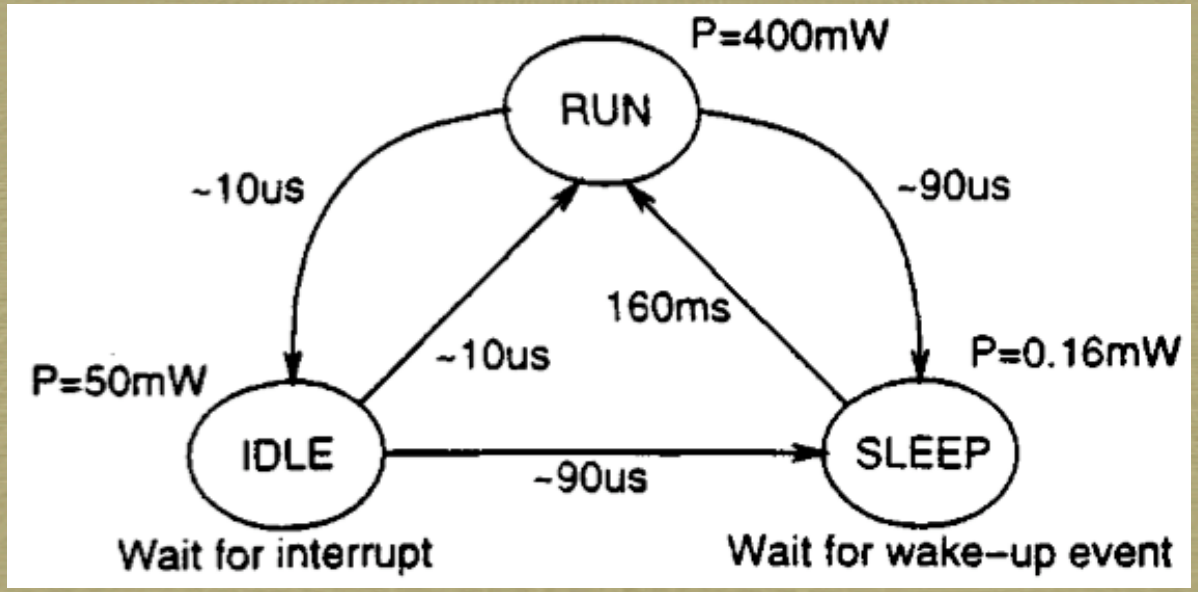
- A design methodology to dynamically reconfigure a system to provide services with minimum number of active components
- Assumptions:
  - Non-uniform load on system
  - Prediction of load is possible
  - Observation/prediction does not consume too much power

DPM system



- Power manager implements a policy
  - E.g. timeout - shut down unit if idle for a some time

Power state machine

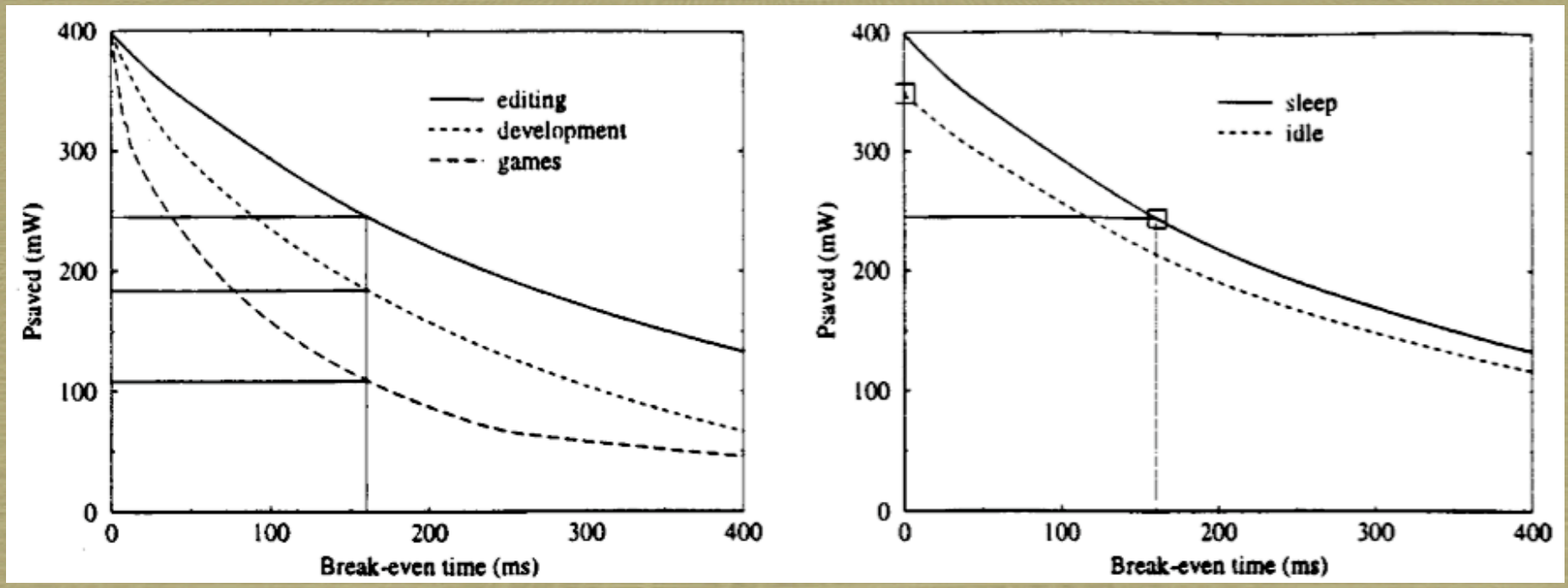


- Power managed components have a number of modes of operation
  - Looks like a state machine
- Transitions have a cost (usu delay)

When to switch state?

- DPM must decide when it is worthwhile to switch to a low-power state and to which one
  - Speed and energy must be considered
- *Break-even time*: minimum inactivity time required to compensate the cost of state switching
  - For each low-power state
  - States with lower break-even times are easier to be exploited by DPM
- Performance loss tolerated?
  - No: inactivity time <= idle time
  - Yes: inactivity time can be longer

Break-even time vs power



- The exploitability of a state depends on both the characteristics of the state and the those of the workload

Power policy optimisation

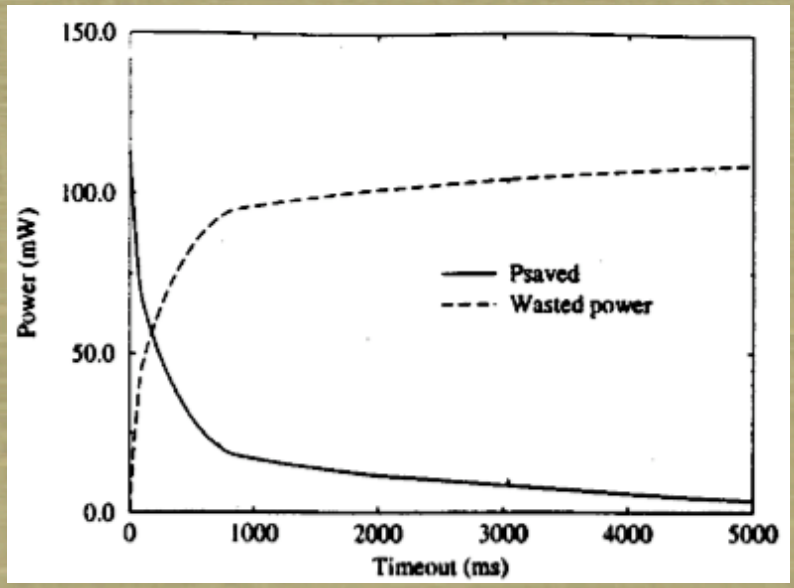
- Predictive techniques
  - E.g. time-out
  - Can be static or adaptive
  - Typically 2 states only (running, idle/sleep)
- Stochastic techniques
  - Using Markov models
  - Allows policies with more than 2 states

Predictive techniques

- Assume correlation between history and near future of workload
- Overprediction - predicted idle times are longer than reality - perf penalty
  - Similarly underprediction - power waste
- Safety - complement of risk of making an overprediction
- Efficiency - complement of risk of making an underprediction

Time out (fixed)

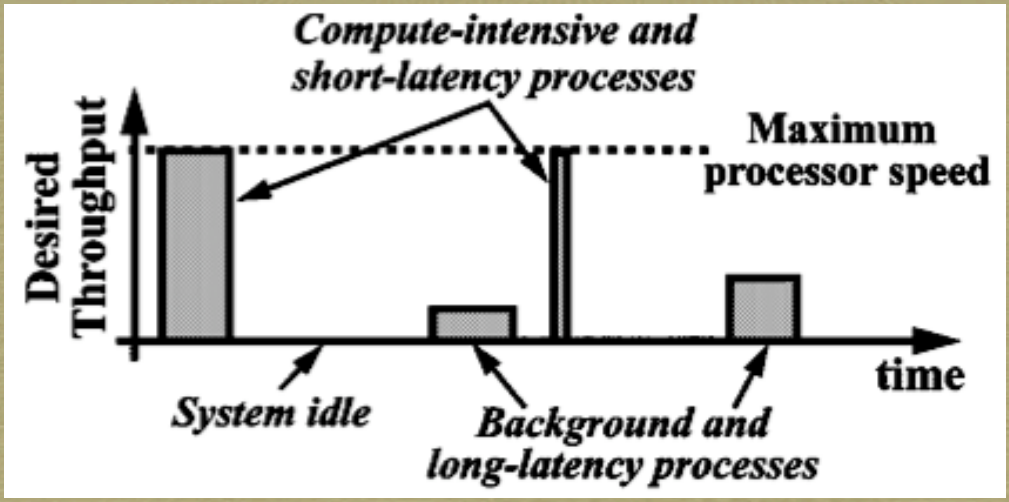
- If idle for T_{TO}, put in low-power state
- Assumes prob. $P(T_{idle} > T_{TO} + T_{BE} \mid T_{idle} > T_{TO}) \approx 1$
- Advantages:
  - Safety can improve with longer TTO
- Disadvantages:
  - Efficiency is traded-off for safety



Time out problems

- Power wasted before timeout expires
  - Solution: Predictive shutdown
  - E.g. in interactive graphic terminals, short busy periods are followed by long idle times. If prev busy time < threshold, shutdown immediately when idle
- Performance penalty always paid when idle time finishes
  - Solution: Predictive wake-up

DVFS



- Processor can scale voltage-frequency
  - Must be able to operate at top throughput for compute intensive tasks
  - and conserve energy for other tasks
  - OS module sets required frequency, regulator (circuit) finds the corresponding supply voltage

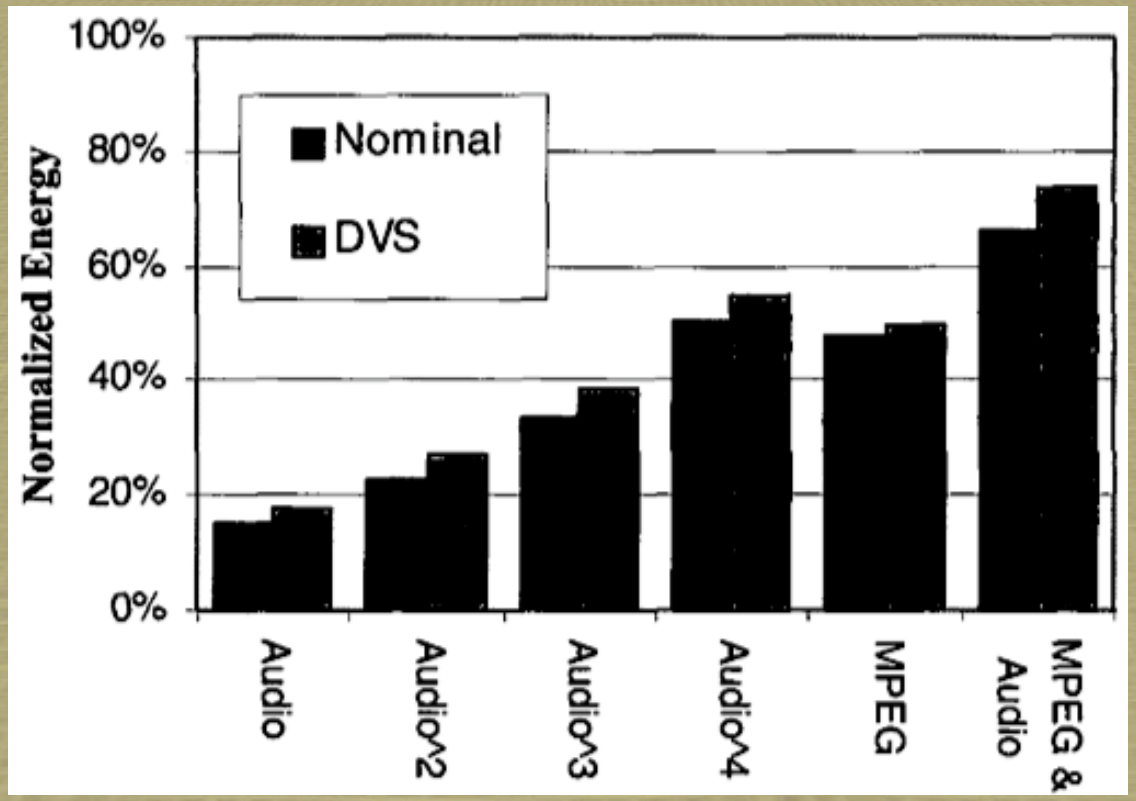
Voltage scheduling

- Part of the OS, determines the CPU freq and indirectly the supply voltage
- Normal, temporal scheduling still there
  - Decides which task runs next using an earliest deadline first (EDF) policy
- Three classes of applications
  - High-priority - ignored by voltage scheduler
  - Deadline-based - application provides completion deadline (for current *frame*)
  - Rate-based - automatically converted to deadline-based
- Applications may miss deadlines
  - They must handle this *gracefully* (add buffers, etc)

Scheduling algorithm

- Set the speed to $\max_{\forall i \le n} \left(\frac{\sum_{j \le i} work_j}{deadline_i - currenttime} \right)$
- O(n) time
- Includes future (known) tasks so as to allocate time for them
- Reevaluated for each new task and at deadlines
  - So it only needs to get the next speed right

DVFS results



- Nominal - fixed constant speed (with full knowledge of workload)

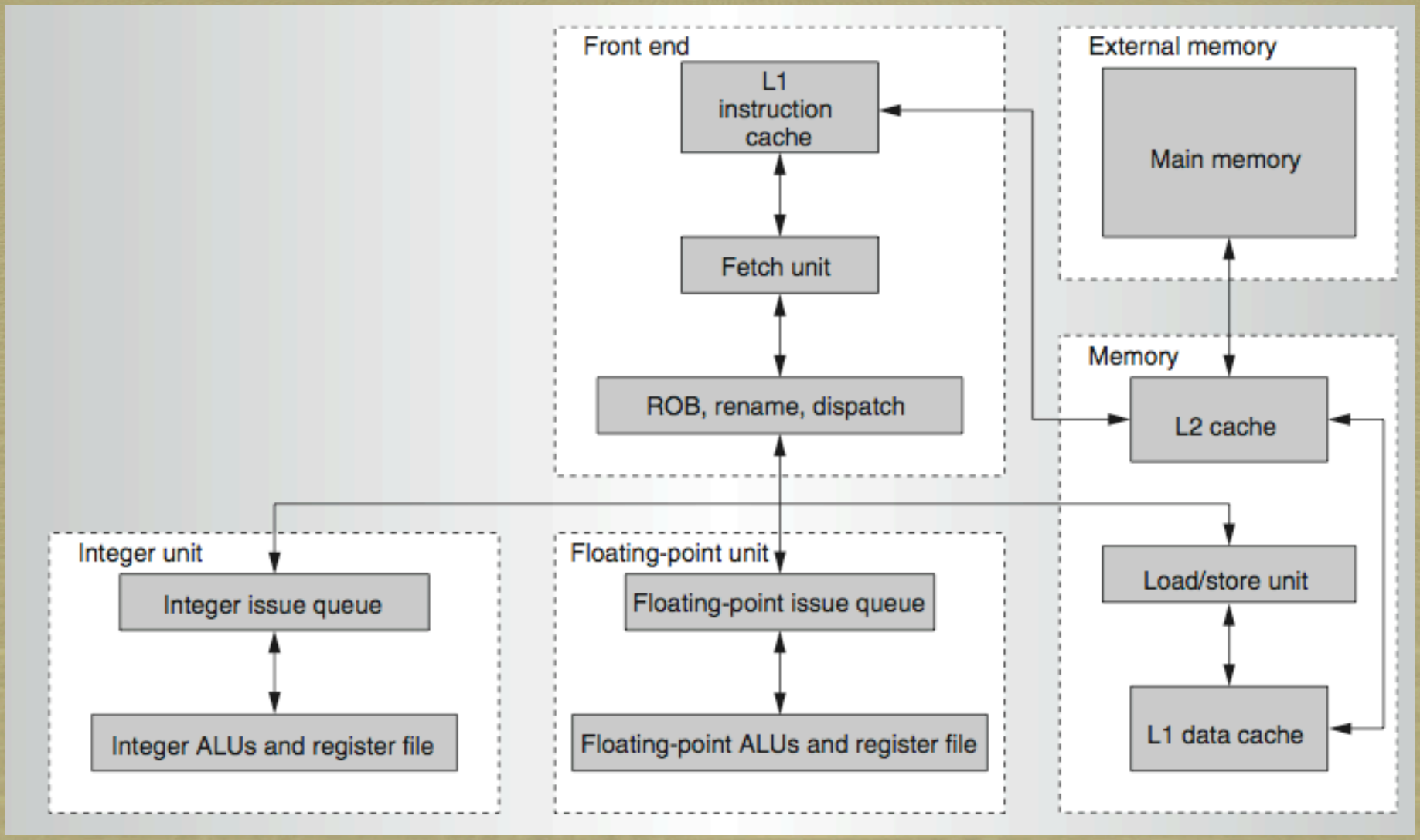
Multiple clock domain(MCD) u-arch

- Controlling clock skew in a large chip is difficult
- Globally Asynchronous Locally Synchronous (GALS)
  - Allows multiple clock domains in a chip
  - Each block is synchronous
  - Communication is asynchronous
- Magklis et al (Micro 03) propose applying DVS independently in each domain

Partitioning the domains

- Explicit synchronisation required for communication between domains
- Choose domains with relatively little communication
- Choose domains which communicate via buffers
  - Buffers are required for synchronisation anyway

An MCD processor



On-line control

- Observe buffer occupancy
  - If near full speed-up, need to go faster
- Interval (10K instr) based attack/decay algorithm
  - Attack: if change in occupancy by 1.75%, change speed by 6% in the same direction
  - Decay: if no change or no activity, drop speed by 0.175%
  - If IPC changes by more than 2.5%, don't change - helps prevent nasty inter-domain effects
  - Force an attack if no change for 10 intervals in a row

Profiling-based control

- Profile application to find long-enough running code
- Generate dependency graph
- Find slack time and re-distribute it by slowing down code fragments out of critical path
- Instrument code with special instructions to trigger speed changes

MCD-DVS evaluation

- Using modified Simplescalar/wattch
- Average performance degradation 7%
- Online control 17% average EDP reduction
- Profiling based control 27% average EDP reduction