Tradeoffs between complexity, power and performance

Victor Zyuban
IBM T.J. Watson Research Center

Performance, power and complexity

- Complexity is not proportional to area
  - Data flow is responsible for most of the area, control circuits which typically take very little area are responsible for most of the complexity
  - Arrays take significant area but logically are very simple
- Complexity is not proportional to power
  - Data flow with lots of activity is responsible for most of the power
  - Arrays and register files are logically simple, but are a significant power component
  - Control circuits that take care of corner cases show very little switching activity but are the most challenging in terms of complexity
  - Hardware for the handling of memory coherence, translation, MP support, microcode typically shows very little activity, but is most challenging in terms of the number of bugs
- Performance always costs complexity

Trading complexity for power

- Clock gating
  - Saves switching power, but increases design complexity
  - In certain timing critical sections clock gating requires redesign
    - when asynchronous even is expected, some prediction hardware may be needed to speculatively turn on the hardware in anticipation of the event
- Adaptive structures, reconfigurable structures (proposals from academia)
  - Most architectural power-savings techniques proposed by academia are not adopted by industry because of complexity

Trading complexity for area

- Multi-cycle issue to functional unit saves area are but increases complexity
- Techniques to save area: sharing ports to register files, sharing register mappers, sharing some of the functional block between units all lead to growth in complexity
- Multithreading: sharing structures between threads saves area but increases design complexity
  - Sharing execution units, branch prediction
  - Sharing caches, tlbs, mcode, decode, interrupt handling
  - Sharing issue windows, load-store queues, issue logic

Trading performance for complexity

- Pipeline depth increases frequency and drives the complexity
  - Need to precompute control signals
- Pipelining of the state machines beyond certain point may be extremely challenging.
- Stalling of the pipeline requires overflow buffers
- Most of the architectural performance features drive complexity
  - Out of order issue, speculative issue, replacing stall with rejects and instruction replays, run ahead execution

Some thought on the complexity metric

- The tradeoff space looks more like this: performance and complexity versus power and area.
- Improving performance and reducing power will inevitably drive design complexity. I don't think there is a way to change this.
- What we can do is use sensitivity analysis to target complexity balanced design.

  In a complexity balanced design the marginal complexity cost of every performance improving and power saving features are the same

  Suppose we are at the design point where we are trading 3% power per 1% in performance

  A certain performance feature improves performance by p% to compensate for increase in complexity we would have to give up certain power savings features (say some of clock gating), which would increase power by q%. The performance feature is justified if 3p>q.

  If on top of that the performance feature has a direct power costs x%, then it is justified only if 3p>q+x