SoC Chapter 4: System Design

Modern SoC Design on Arm
End-of-Chapter Exercises

Q1 Speedup

If an accelerator multiplies the performance of one quarter of a task by a factor of four, what is the overall speedup?

Answer:

This is a simple application of Amdahl's Law. The performance was $4/(1+1+1+1)$ and is now $4/(1+1+1+0.25)$ so it has increased by 23 percent.

Q2 Queuing Theory

The server for a queue has a deterministic response time of 1 µs. If arrivals are random and the server is loaded to 70% utilisation, what is the average time spent waiting in the queue?

Answer:

The utilisation, ρ, is 0.7. The server is operating at 1 MHz since its response time is $1/\mu = 1$ µs. The queue delay for M/D/1 is

$$W_q = \frac{\rho}{2\mu(1-\rho)} = \frac{0.7}{0.6} \text{ microseconds}$$

Which is 1.17 microseconds. Not asked for: the total 'system time' must have the service time added on, which will be 2.17 microseconds.

Q3 Queuing Theory — Two Queues

If the server is still loaded to 70% but now has two queues, with one being served in preference to the other, and 10% of the traffic is in the high-priority queue, how much faster is the higher-priority work served than the previous design where it shared its queue with all forms of traffic?

Answer:

The precise answer to this question involves solving the balance equations for a 2-D embedded Markov diagram based on the PASTA (Poisson arrivals see time averages) theorem. Answering this way requires more advanced Queuing Theory than is presented in the book.

Note that the high-priority traffic will be unaffected by the low priority traffic, except for when it gets to the head of its queue if the server is busy with a low-priority item. In that case, we see a small amount of *priority crosstalk*. An approximate (engineering) approach is to assume that the high priority traffic experiences the queuing delay experienced by a server loaded by just itself ($\rho = 0.07$) plus the mean residual life of the server time when the server is busy with a low-priority work item. This queuing time is

$$W_{q\,hi} = \frac{\rho}{2\mu(1-\rho)} = \frac{0.07}{2 \cdot 0.93} = 0.038 \text{ microseconds}$$

which is almost negligible compared with the 1 μs service time it is about to experience for itself. The mean residual life of the low-priority traffic datum is more significant. The server is busy for a microsecond with a low-priority item with probability 0.63 and the mean residual life of a uniform distribution is half the service time, so this will add on 0.31 microseconds on average. The theoretical error in this engineering approach arises since the high-priority queue is not seeing deterministic response time from the server, but this error is likely to be negligible compared with that arising from assumption of independent Poisson (random) arrivals, etc..

Q4 Energy Saving

If a switched-on region of logic has an average static to dynamic power use of 1 to 4 and a clock gating can save 85% of the dynamic power, discuss whether there is a further benefit to power gating.

Answer:

A power-gated region will have very little leakage current when disabled provided the power gating transistors properly turn off. Given an on-to-off resistance ratio for the gating transistors of at least 10^4, which is easy to achieve with conventional MOSFETs, the off-leakage energy will typically be less than 0.01 percent of the region's active energy use. Clock gating is saving most of the active power but will not affect the static power, so there is considerable further power saving available from power gating.

The ratio of static to dynamic power use given suggests the design is implemented in a comparatively high leakage technology (or at least has a fair number of such transistors). The power gating transistors should have a different doping level, gate bias or other structural variation to achieve a suitably low off current.

Power gating has associated overheads in terms of power distribution network complexity, the area of the gating transistors and their controllers. Any increase in chip area will extend the length of the interconnect, resulting in extra power use. If the region in question is nearly always in use, there is no benefit from power gating. If it is seldom used, relatively large and it is relatively obvious when it should be turned off, then it is probably worthwhile.

Q5 Debug Trace

What is the minimum information that needs to be stored in a processor trace buffer to capture all aspects of the behaviour of a program model given that the machine code image is also available?

Answer:

To fully trace a deterministic subsystem, like a processor core, either the data externally entering it needs to be logged or else the data it internally deletes needs to be logged. Or any valid hybrid. Data is internally deleted when a register is loaded with a new value, including the program counter when it transfers to a code label with more than one antecedent.

Given that an accurate copy of the machine code has been provided to the trace analyser and that the hardware is working correctly, all of the intermediate states can be inferred from the log.

With faulty hardware, the intermediate state trajectory may be wrong and the source of the bug might not be inferable from any given log. On the other hand, to debug faulty hardware, a large number of test programs can be logged in separate runs and hopefully the problem becomes clear.

Non-deterministic behaviour is the hardest to debug, especially when the occurrence of the fault is rare. It is then handy to have detailed traces containing information that would be redundant on properly working hardware. In these situations, an investment in trace buffer real estate really pays off.

Q6 Fault Redundancy

A 100-kbit SRAM mitigates against a manufacturing fault using redundancy. Compute the percentage overhead for a specific design approach of your own choosing. Assuming at most one fault per die, which may or may not lie in an SRAM region, how do the advantages of your approach vary according to the percentage of the die that is an SRAM protected in this way?

Answer:

A square array of 10^5 bits would have 316 columns each of 316 bits. A practical RAM might have 256 columns of 512 bits. The question is directing us to consider adding an extra row or column of bits to each RAM and 'zapping' this into action during production test to replace a faulty row or column. The zapping could use any technology, such as write-once floating-gate PROM cells.

Adding one row or column will add about 0.3 percent overhead to the RAM cell array. The additional logic to swap a faulty row or column with the spare one will increase the RAM access delay. It is likely best to add the complexity to the column multiplexors instead of the row binary-to-unary decoders since the column logic can act in parallel with the row activation.

It became practical to put a 100-kbit (e.g., 32Kx8 bytewide) on a single chip in the mid-1980s and this is a small RAM by today's standards. Another approach to SRAM reliability is forward error correction using ECC. Using ECC is common practice for large RAM areas and can handle random (noise or radiation) upsets as easily as fixed-site faults. The Shannon-Hartley theorem reveals that the data overhead of ECC tends to zero as more data is covered. Equally, the overhead of the ECC logic, which is needed only once, becomes ever better amortised. So ECC becomes more efficient than adding an overhead proportional the square-root of the number of bit cells. The downside of super-efficient ECC is latency owing to a big block having to be processed to reach that efficiency.

Q7 High-level Energy Modelling

Assuming an embarrassingly parallel problem, in which all data can be held close to the processing element that operates on it, use Pollack's rule and other equations to derive a formula for approximate total power and energy use with a varying number of cores and various clock frequencies within a given silicon area.

Answer:

An embarrassingly parallel problem has zero inter-task dependencies and should speed up linearly with the number of processing elements or cores. Pollack's rule of thumb suggests the area and energy cost of a processor core is quadratic in its IPC. Let N_c be the number of cores, f be the clock frequency, I_{pc} be the instructions per clock for the chosen cores and T be the total number of instructions needed for the task (assumed constant regardless of number of cores).

We must also model the number of cores that will fit in our given area, which we will define as N_0 (e.g., about 100) when the IPC is unity (e.g., simple 5-stage RISC pipeline).

The number of processors available is: $N_c = N_0/I^2_{pc}$

The time to complete the task is: tasktime $= T/(f \cdot N_c \cdot I_{pc}) = (T \cdot I_{pc})/(f \cdot N_0)$.

The energy of computation is: $E_{\text{total}} \propto T \cdot I^2_{pc}$

The power is:

$$P = E_{\text{total}}/\text{tasktime} \propto T \cdot I^2_{pc} \cdot \frac{f \cdot N_0}{T \cdot I_{pc}} = I_{pc} \cdot f \cdot N_0$$

If DVFS is possible in the available technology, then energy use grows quadratically with the clock frequency and that effect should be included too.

Q8 Static Scheduling

Consider a succession of matrix multiplications, as performed by *convolutional neural networks (CNNs)* and similar applications in which the output of one stage is the input to the next. Is FIFO storage needed between stages and if so, could a region of scratchpad RAM be sensibly used or would it be better to have a full hardware FIFO buffer?

Answer:

The succession of matrix multiplications will typically not all have the same dimensions (CNNs used for image recognition operate on successively smaller arrays as more about the image is discovered). Hence there is a need for data rate adaptation between the successive stages of processing. This might suggest FIFOs are suitable, but not so in reality. [NB: there are other types of operator beyond matrix multiplication, such as feedforward and non-linear stages, but these are less computationally expensive.]

A vast number of designs for Neural network accelerators have been published. Many designs use DRAM for data and on-chip SRAM for coefficients. Data has to move backwards and forwards to DRAM many times. Recent designs have eliminated most of the copying to DRAM by making very large chips and putting enormous amounts of SRAM on that chip (e.g., the Groq Tensor Streaming Processor).

A succession of matrix multiplications has no data-dependent control flow. It is highly amenable to offline static scheduling, as used in high level synthesis (HLS) and systolic arrays. Designs that minimise the controller overhead will have more area available for SRAM instead.

The term ’full hardware FIFO’ probably refers to an on-chip component with its own circular buffer memory and input and output pointers. The word width can be any custom size on-chip, so an answer which mentions that a lot of FIFOs would be needed owing to the quantity of data or that it is not efficient to have a lot of small memories and lock-step replicated state in the FIFO control logic is specious. However, there would be replicated state between, say, the input counter of such a FIFO and the output stage of the matrix multiplier that was feeding it. Replication is unnecessary. Indeed, the clock-by-clock depth of data in the FIFO should be knowable at compile time as part of the static schedule, so having separate input and output pointers may not be helpful.

Where data crosses clock domains or encounters variable-latency operations (especially transfer to/from DRAM), FIFOs are appropriate, but the addresses for all the principal memory operations are computable by the application compiler and the memory is being used in the same way as ‘scratchpad’.

Address computation at run-time will still be happening, but merely as an aspect of ‘decompressing’ a concisely-stored static schedule. CNNs with sparse coefficients require more elaborate decompression than just counting.