## 4. Sample Performance Analysis

This section is intended to describe a fairly simple performance analysis of a system with various L2 cache options. It uses the reference implementation of the PowerPC Reference Platform Specification as the base system. Some simplifying assumptions are made, but the point is to demonstrate a first approximation technique at performance analysis. It will also demonstrate the degrees of performance improvement when both write-through and copy-back caches are added to a system. It is extremely important for system designers to understand the workloads they expect to run and the demands these workloads will place on the memory subsystem. Only then can the appropriate L2 cache design be properly determined.

The PowerPC Reference Platform Specification has an upgrade socket which can be used to add a look-aside L2 cache. Let us assume that we are considering implementing either a write-through or copy-back design, and need to understand the benefit of each.

What we know about the reference implementation:

- The processor and processor bus run at 66Mhz
- Memory accesses take 25 bus cycles on average (10-5-5-5) to complete
- Fetches take an additional 3 cycles in the processor to complete

What we will assume:

- Typical workloads will generate an average of 9 fetches to memory / 100 instructions
- Typical workloads will generate an average of 2 writes to memory / 100 instructions
- Our infinite L1 cache CPI (cycles per instruction) is approximately one (infinite L1 cache means all program fetches and stores hit in the L1 cache).

Given these assumptions, we can bound performance (using CPI as the metric) for the system with no L2 cache. In the worst case, if no instructions can be executed 'under' a memory access, then it would take 404 cycles to complete the 100 instructions. This is calculated by assuming each instruction takes one cycle once its data is present in the L1, and the fetch miss penalty is 28 cycles (25 bus and 3 processor), while the store miss penalty is 26 cycles (25 bus and 1 processor). Thus we have 100 + (28 * 9) + (26 * 2) = 404. CPI then equals 404 cycles divided by 100 instructions, or 4.04.

Assuming the best case for the above, then as much instruction processing as possible would take place under memory accesses. If the L1 cache misses were evenly spread throughout the 100 instructions, then the 100 cycles to calculate the instructions once data is present in L1 can be hidden under the 304 cycles taken up by memory accesses. This results in a best case CPI of 3.04. This is theoretically possible because the 601 processor is bypassed fetch data after the first two double words are returned. This means that there are 10 cycles at the trailing edge of each fetch for which no data dependencies exist. For stores to memory, no data dependencies exist. This means that for the 100 instructions, 9*10 + 2*26 = 142 cycles are available to execute instructions.

In reality, the CPI will be somewhere in between. Memory accesses won't occur regularly, and many instructions will be executed under them. For a better approximation, the memory subsystem can be modeled as a single server queuing system with a deterministic service time, sometimes referred to as an M/D/1 model. We will assume the distribution of memory accesses among the instruction stream is random (Poisson process). Given this, the following holds:

$$C = C_\infty + C_w + C_q \text{ where:}$$

$C$ = The system cycles per instruction, or CPI.

$C_\infty$ = The infinite L1 cache CPI for systems without an L2 cache, or the infinite L2 cache CPI for systems with an L2.

$C_w$ = The average CPI service time for the memory access and data return of a fetch request, and

$C_q$ = The average CPI spent waiting for the memory to become available.

These CPI terms can be further defined as follows:

$$C_\infty = C_{\infty L1} + M_{L1} * (C_{L2fd} - C_{\infty L1} * n) \text{ where :}$$

$C_{\infty L1}$ is the infinite L1 cache CPI,

$M_{L1}$ is the L1 fetch misses per instruction,

$C_{L2fd}$ is the delay in cycles per fetch for first data to be returned from the L2 to the processor, and

$n$ is the number of instructions per L1 fetch miss for which no data dependency exists, and thus can be executed 'under' the L2 fetch.

$$C_w = M_{L2} * (C_{Mfd} - C_{\infty L1} * n) \text{ where:}$$

$M_{L2}$ is the L2 fetch misses per instruction, which is just $M_{L1}$ multiplied by the L2 miss ratio, and

$C_{Mfd}$ is the delay in cycles per fetch for first data to be returned from memory to the processor.

$$C_q = \frac{1}{2*C*(\frac{\mu}{\lambda})^2 - 2*(\frac{\mu}{\lambda})} \text{ where:}$$

$C$ is the system CPI (this is what we're solving for!),

$u$ is the service rate of the memory in accesses per cycle, from the point of view of the processor. Since the processor is stalled after executing $n$ instructions once a fetch is initiated, we must calculate this rate deducting the cycles that the processor is stalled from the total cycles to execute a fetch. This results in a higher service rate than if the processor never stalled. Finally,

$\lambda$ is the arrival rate to the memory in accesses per instruction, and can be calculated as the L2 fetch misses per instruction plus the L2 store misses per instruction.

This may seem difficult but it simplifies somewhat with certain assumptions. First, we already assumed the infinite cache CPI $(C_{\infty L1})$ to be one. Next, we will assume that n, the number of instructions per L1 fetch miss for which no data dependency exists also to be one. This basically means that for every L1 fetch miss, the compiler is smart enough to find an average of one instruction which can be executed before the missed data is returned. The fact that $C_q$ has the system CPI ($C$) in its denominator results in a quadratic equation when solving for $C$.

### 4.1 System with no L2 Cache

For a system with no L2 cache, the following is true:

$C_\infty = C_{\infty L1} = 1$;

$M_{L2} = M_{L1} = .09$;

$C_{Mfd} = 18$, since the 'first data' for the 601 requires two beats of data and three cycles are used in the processor when fetching, so

$C_w = .09 * (18 - 1 * 1) = 1.53$;

$u = \frac{11}{9*(25-15+1)+2*25} = .0738$; Note the denominator, where the fetch access is reduced by the cycles spent retrieving first data plus n, the number of instructions for which no data dependency exists.

$\lambda = .09 + .02 = .11$, since there are 9 L1 fetch misses per 100 instructions and 2 L1 store misses per 100 instructions.

$\frac{\mu}{\lambda} = \frac{.0738}{.11} = .671$;

So we are left with the equation:

$$C = 1 + 1.53 + \frac{1}{2*C*(.671)^2 - 2*(.671)};$$

Solving the resulting quadratic equation yields:

$$C = 3.185 \text{ cycles/instruction.}$$

### 4.2 System with a write-through L2

Let us now assume we have a look-aside, write-through L2 cache, with 3-1-1-1 access latency, and a hit ratio of 75%.

For this case, the following is true:

$C_{L2fd} = 7$ since 4 cycles are required to return two beats of data and 3 cycles are spent in the processor;

$C_\infty = 1 + .09(7-1) = 1.54;$

$M_{L2} = M_{L1} * (1 - .75) = .09 * .25 = .0225;$

$C_w = 0.225 * (18 - 1) = .3825;$

$\lambda = .09 * (1 - .75) + .02 = .0425;$

$\frac{\mu}{\lambda} = \frac{.0738}{.0425} = 1.736;$

Therefore, $$C = 1.54 + .3825 + \frac{1}{2*C*(1.736)^2 - 2*(1.736)};$$

Solving, we get: $$C = 2.235 \text{ cycles/instruction.}$$

### 4.3 System with a Copy-back L2

Let us now assume we have a look-aside, copy-back L2 cache, with 3-1-1-1 access latency, and a hit ratio of 75%. The only change from the above is in the arrival rate to the memory subsystem.

$\lambda = (.09 + .02) * .25 = .0275;$

$\frac{\mu}{\lambda} = \frac{.0738}{.0275} = 2.684;$

Therefore, $$C = 1.54 + .3825 + \frac{1}{2*C*(2.684)^2 - 2*(2.684)};$$

Solving, we get: $$C = 2.03 \text{ cycles/instruction.}$$

This analysis shows that for the reference implementation, an improvement of about 30% is achieved for a system with no L2 cache versus a system with an write-through L2 cache. An improvement of about 9% is achieved for a system with copy-back L2 cache versus write-through, other factors being equal. Granted this model is far from perfect. First of all, it assumes no queuing delay in the L2 cache itself. Second, it does not consider I/O accesses into the memory subsystem. Still, it should provide a reasonable expectation of relative performance for various L2 cache options.