Technical Disclosure Commons

Defensive Publications Series

August 16, 2019

Poll-optimized adaptation of PCI-express

N/A

Poll-optimized adaptation of PCI-express

ABSTRACT

CPUs monitor the network interface by spin-loop polling or by the use of interrupts. Spin-loop polling delivers high performance but consumes significant amounts of power and generates heat. The use of interrupts is less power hungry but causes substantial latency.

This disclosure describes techniques that introduce, for certain CPU requests, a bounded (<10 μs or so) delay in the return of a completion packet from the peripheral device to the CPU. This reduces the energy spent by the CPU on read/compare/repeat loops and enables operations like direct memory access to get more bandwidth while retaining the high performance of spin-loop polling. The techniques leverage the multiple-outstanding-transactions and out-of-order completions features of the PCIe (or similar) buses to achieve zero or near-zero delay penalties while substantially mitigating the power and thermal consequences of spin-loop polling.

KEYWORDS

- PCI express
- PCIe
- Spin-loop polling
- Direct memory access (DMA)
- Transaction level packet (TLP)
- Memory read request TLP
- Out-of-order completions
- Multiple outstanding transactions

BACKGROUND

In a typical computer, connections between peripheral devices (also known as input/output or I/O devices) and the processor or memory are made using PCI-express (PCIe) ports on the CPU socket. The two primary mechanisms for CPU software to detect an I/O event, e.g., read, write, send, receive completion, etc., are interrupts and polling.

An interrupt causes an exception on a CPU, where program execution is suspended and the CPU enters a more privileged state to execute code from an exception vector. After the interrupt exception service, the CPU drops to the prior privilege state and resumes pre-interrupt execution. This is a good model for infrequent and unpredictable I/O events. However, some I/O operations are so fast and so predictable that interrupt-driven operation is inefficient and limits performance. The path into the interrupt service handler takes hundreds of CPU clocks, and unwinding after interrupt service is almost as costly. In these cases, polling is often used.

Polling is simple: a CPU spins in a loop, repeatedly reading a register across the I/O port, or in some cases reading a memory location that can be modified by a direct memory access (DMA) cycle from the I/O port, until the value loaded in the register indicates completion. The latency of event discovery is taken down to the level of a memory or memory-mapped I/O load followed by a compare and a branch. The price to be paid for using a polling loop is that the CPU cannot perform other tasks while waiting on the event, and while the processor is in the polling loop, it runs at maximum speed, consuming large amounts of power. Modern, multi-core CPU packages mean that having one core tied up in a polling loop does not amount to a pause in computer operations, as other cores are available to run applications and service exceptions. However, power and thermal budgets also become more significant and complex to manage.

A CPU typically requests data from a peripheral device in the form of a memory read request transaction level packet (TLP). Under conventional polling behavior, the peripheral marshals the requested data into a completion TLP as quickly as possible and sends it to the CPU. If the looked-for status is not reflected in the data, the CPU loops back and immediately tries another CPU load from the PCIe port, which in turn results in another read, another completion TLP, and another test of the returned data. Conventional polling thus results in the CPU entering a busy-wait mode, burning significant amounts of power. Over a large fleet of computers, e.g., in data centers, the amount of power thus expended can easily be in megawatts.

DESCRIPTION

Per the techniques of this disclosure, a modification of peripheral device behavior is made at the low level of the interconnect to reduce power consumption and heat dissipation in open-loop polling, while preserving low-latency response to completion. The techniques apply to interconnect schemes that have the facilities of multiple outstanding transactions and out-of-order completions, e.g., PCI-express.

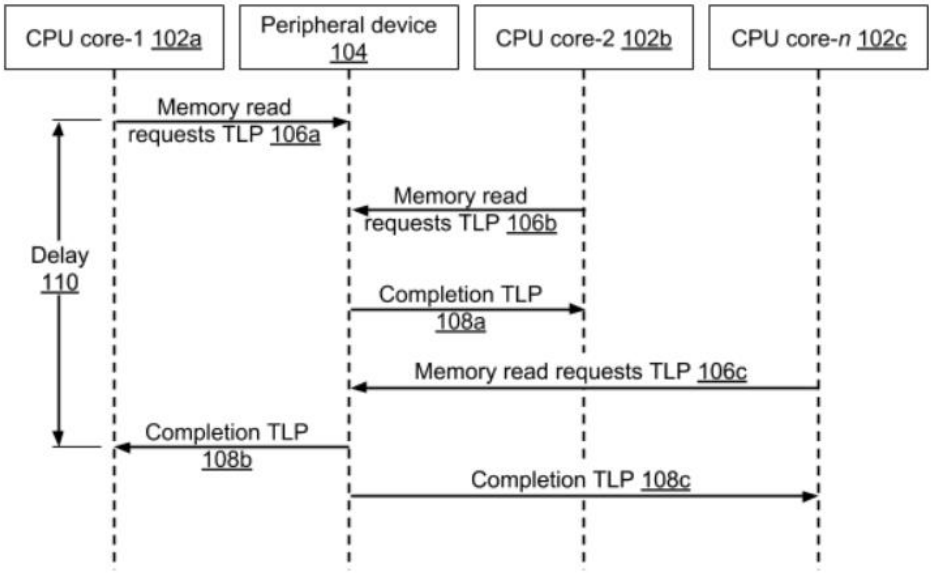

Fig. 1: Poll-optimized communications over a PCIe-like peripheral interconnect

Fig. 1 illustrates poll-optimized communications over a PCIe (or PCIe-like) peripheral interconnect, per techniques of this disclosure. Multiple CPU cores (102a-c) are in communication with a peripheral device (104) over the PCIe. In a polling loop, a CPU core emits at the port interface associated with the peripheral device a transaction level packet (TLP) of type memory read request. For example, in Fig. 1, the peripheral device receives respectively from CPU cores 1, 2 and *n*, the memory read request TLPs 106a, 106b, and 106c.

A TLP has a target address, a data transfer size, a requestor ID, and a transaction ID, also known as a tag. When the peripheral device receives a TLP, it decodes the address to determine the memory or logic state being referenced, and sends back to the requestor a completion TLP. The completion TLP is matched with the request TLP via the tag, which is copied from the request TLP to the completion TLP. For example, in Fig. 1, the peripheral device sends back respectively to CPU cores 1, 2 and *n*, the completion TLPs 108b, 108a and 108c.

Per the properties of PCIe, completion TLPs may be sent out-of-order, and there may be multiple simultaneous outstanding transactions. For example, in Fig. 1, CPU core-1 is the first to send a request TLP (106a) but the second to receive a completion TLP (108b). While CPU core-1 is awaiting a completion TLP, the peripheral device receives request TLPs from CPU cores 2 and *n* (106b-c), and responds to CPU core-2 with a completion TLP (108a). Outstanding transactions are also known as in-flight transactions. In PCIe, there are 32 legal tag values and thus, up to 32 requests may be in flight at the same time.

Per the techniques of this disclosure, a circuit or microcoded sequence is inserted, either in an ASIC implementing the PCIe (or similar interconnect) or as a circuit block between such an ASIC and the connector to the I/O port of the CPU, which causes a delay (110) in the return of completion TLPs of certain request TLPs. Request TLPs that receive delayed completion TLPs

are those that are evaluated as low priority or less interesting. The delay is bounded, e.g., it is of the order of tens of microseconds. Responding in tens of microseconds instead of the current order-of-microseconds reduces the activity on the PCIe port, allowing other operations like DMA to get more bandwidth, and reducing the energy being spent by the CPU core on read/compare/repeat loops.

If the priority of the memory read request is high, then its completion TLP is not delayed. If the priority of the memory read request TLP becomes higher (more interesting) during the delay period, then the delay is aborted and its completion TLP is sent immediately. Under these conditions, optimized PCIe polling, disclosed herein, exhibits the same responsiveness as baseline polling while operating more efficiently.

Determining if a delay is to be applied

A memory read request can be encoded in various ways to determine if a delay is to be applied to its completion TLP. Example techniques to determine the applicability of delay are as follows:

- Certain memory-mapped status registers can be treated as always being delay-worthy without imposing any further protocol. In this regard, PCIe writes to other control registers can be used to robustly configure the polled-status behavior of status registers.
- In a form of register-address aliasing, a modulated-delay poll response from the register at address 0x0000XXXX can be obtained by sending a read request for address 0x0001XXXX.

Determining the length of the delay

In an example, the length of the delay can be determined as follows. Each time a delayed-load poll is performed, the value returned is stored, along with the target address, in a history-

buffer register associated with the PCIe logic. This value is set to some invalid state on reset, and is likewise invalidated when a delayed-load poll is sent to an address different from the last. When a delayed-load poll is received, the data at the target address is compared against the history buffer. If they differ — because the old value was invalid (e.g., first poll in a loop), or because the value has changed from the last poll — a completion TLP with the data is sent immediately. If the history is valid and the new value is unchanged from history, the response is delayed by the configured or designed-in amount of time before being returned. In the case of an ASIC where the PCIe control logic managing the packets and the history is integrated with the peripheral functionality, the status-register logic can signal a value change (or a potential value change) to the delayed-load logic to abort the timeout and transmit the new/current value in a prompt completion TLP.

If the ASIC implementing PCIe is a black box, e.g., with little internal visibility, the technique for determining the length of the delay can be applied using a layer of logic between the ASIC and the I/O port. The layer of logic, also known as a shim layer, contains the history buffer and associated logic as described above. It detects memory read transactions that are to be treated as delayed-load poll events. An event not identified as a delayed-load poll event is passed through the shim to the PCIe interface of the ASIC almost immediately, e.g., with the minimal delay associated with the time to de-serialize and compare the address in the TLP.

If the TLP is a read request to a poll target, the shim layer passes on the request to the ASIC and also monitors the returning TLP stream for the completion response with the same tag value, buffering (and thus delaying) the responses long enough to de-serialize and identify these. Non-poll completions are sent on with no further delay, but completions with the poll tag have their data payload checked against the history buffer in the shim. Differing values signal a

change, and the completion is sent on to the CPU.

If the values are the same, the shim logic generates a new TLP identical to the original poll (including using a tag identical to the original poll) and sends it into the ASIC at the next available opportunity. The next available opportunity is generally when the input filter buffer is empty, or if the input filter buffer is one read request TLP deep. If the CPU starts sending a new read, write, or other TLP to the device, the shim has enough time to push the re-poll to the ASIC before the input filter buffer overflows. This adds constant delay while the shim is active, but avoids the need to interact with the PCIe flow-control protocols. For devices whose on-chip PCIe behavior is controlled by microcode, delay penalties are thus kept to zero or very low, while substantially mitigating the power and thermal consequences of open-loop polling.

CONCLUSION

This disclosure describes techniques that introduce, for certain CPU requests, a bounded (<10 μs or so) delay in the return of a completion packet from the peripheral device to the CPU. This reduces the energy spent by the CPU on read/compare/repeat loops and enables operations like direct memory access to get more bandwidth while retaining the high performance of spin-loop polling. The techniques leverage the multiple-outstanding-transactions and out-of-order completions features of the PCIe (or similar) buses to achieve zero or near-zero delay penalties while substantially mitigating the power and thermal consequences of spin-loop polling.