2013 21st Annual International IEEE Symposium on Field-Programmable Custom Computing Machines

Memory Access Scheduling on the Convey HC-1

Zheming Jin
Dept. of Computer Science and Engineering
University of South Carolina
Columbia, SC USA
jinz@email.sc.edu

Jason D. Bakos
Dept. of Computer Science and Engineering
University of South Carolina
Columbia, SC USA
jbakos@cse.sc.edu

***Abstract*—In this paper we describe a technique for scheduling memory accesses to improve effective memory bandwidth on the Convey HC-1 platform.**

In general, when performing a series of accesses to a multibank DRAM, the latencies experienced by the requests may be non-uniform and depend on the pattern of access addresses. This is due to timing requirements of the individual DRAM banks and the memory controller's ability to overlap accesses to different banks. The memory controller on the Convey HC-*x* coprocessor attempts to hide the latency of DRAM accesses by supporting hundreds of inflight memory requests and reordering them in order to move requests to idle banks ahead of requests that are waiting on busy banks. This causes load requests to be returned to the user logic in a different order than originally requested. In addition, the memory controller will stall the user logic's ability to make further memory requests whenever the number of outstanding memory requests exceeds the size of the controller's internal request tracking table. This backlogging occurs when Convey's scheduler fails to hide all the request latencies, either due to some type of limitation in Convey's memory scheduler (which is unspecified since Convey doesn't disclose the design of their memory controller) or caused by the kernel design not uniformly distributing its requests over all available banks.

For memory-bound kernels, the user logic will attempt to access memory on every clock cycle. Any cycle where the logic does not read memory is either due to inefficiency in the kernel's memory interface or due to a stall request from Convey's memory controller. We compute memory efficiency as: *efficiency = ac / ec,* where *ac* = number of access cycles and *ec* = number of observed execution cycles. The number of observed execution cycles includes both the stall cycles requested by the Convey memory controller and the memory access cycles. Our objective is to develop a general purpose kernel memory interface for memory-bound kernels that have a regular access pattern. Our interface reorders the memory references between the user logic and the Convey memory controller. Since the Convey memory controller itself will again reorder the references, our memory interface will buffer the returned data and return it to the user logic in the order that it expects. In other words, we consider three behavioral entities, the kernel (K), Convey's memory controller (C), and our proposed kernel memory interface (I). There are also three memory access orderings: (1) load requests from the interface to memory controller (O_{IC}), (2) returned data from controller to interface (O_{CI}), and returned input data from the interface to kernel (O_{IK}). O_{IC} should be optimized for the memory controller, O_{CI} will be determined by the memory controller (and outside our control), and O_{IK} should be optimized for the kernel logic. The objective of our proposed interface is to: (1) request the kernel's input data from the memory controller in an order that maximizes effective memory bandwidth (which in general is from consecutive addresses, if possible), and (2) buffer the returned data from the controller and send it to the kernel in an order that is optimized for the kernel design.

As a representative kernel, we consider a six-point 3D stencil kernel. We assume that the kernel is fully pipelined and can accept incoming data at the memory's peak throughput, one word per cycle. We assume that the kernel calculates the value of each cell of a 3D space as a function of the values in the cell above, below, to the left, to the right, in front, and behind of the cell whose value is being computed. The kernel expects these input values to arrive at its input ports in consecutive cycles but these values are not stored in consecutive locations in memory. Thus without the proposed interface, all input values would be read from nonconsecutive memory locations and result in poor memory efficiency. In our proposed approach, we designed an address generator that requests data from the memory controller in consecutive order (O_{IC}). Then, we modified Convey's own memory response reorder buffer to buffer the returned values (in the order O_{CI}), and send them to the kernel in the order that they are expected (O_{IK}). Table I shows our results for a 512-entry buffer using a block size of 85, 17, and 8. Using this technique we demonstrate a 33% improvement in effective memory bandwidth.

TABLE I. PERFORMANCE RESULTS FOR STATIC SCHEDULER (X=Y=Z=256)

Reorder buffer	Convey	Proposed	Proposed	Proposed
O_{IC}	Kernel	Consec.	Consec.	Consec.
O_{IK}	Kernel	Kernel	Kernel	Kernel
Block size	1	85	17	8
Efficiency	67%	13%	89%	89%

978-0-7695-4969-9/13 $26.00 © 2013 IEEE
DOI 10.1109/FCCM.2013.55