C3D: Mitigating the NUMA Bottleneck via Coherent DRAM Caches

Detailed Protocol Specification

Cheng-Chieh Huang[1], Rakesh Kumar[1], Marco Elver[1], Boris Grot[1] and Vijay Nagarajan[1]

[1]University of Edinburgh

August 12, 2016

1 Introduction

This document provides a detailed description of the C3D cache coherence protocol [HKE+16].

2 Controllers

The following summarizes the notation and assumptions used in the transition tables:

- Format of receive message action: source?Message
- Format of send message action: destination!Message
- Only the message types Data and PutX carry data.
- No ordering constraints on interconnect.
- The sharer list or owner entry per line b.so points to sockets. The functions llc(socket) and dram(socket) are used to refer to the LLC or DRAM cache controller within the socket respectively; self refers to the controller's socket.

2.1 LLC Controller

Table 1 shows the last level on-chip cache controller.

2.2 DRAM Cache Controller

Table 2 shows the DRAM cache controller.

2.3 Directory Controller

Table 3 shows the main memory directory controller.

Table 1: LLC Cache Controller

	Read	Write	Replacement	src?Data	src?Downgrade	src?Inv	src?PutAck	src?UpgradeAck
I	dram(self)!GetS; → IS	dram(self)!GetX; → IM				src!InvAck;	src!InvAck;	
IS	stall;	stall;	stall;	copy data; hit; → S		src!InvAck; → IS_I	src!InvAck; → IS_I	
IS_I	stall;	stall;	stall;	hit; → I		src!InvAck;		
IM	stall;	stall;	stall;	copy data; hit; dir!DataAck; → M	src!DowngradeAck; → IM_S	src!InvAck;	src!InvAck;	
IM_S	stall;	stall;	stall;	copy data; hit; dram(self)!PutX; → MS;		src!InvAck;		
S	hit;	dram(self)!Upgrade; → SM	→ I			src!InvAck;		
SM	stall;	stall;	stall;	copy data; hit; dir!DataAck; → M		src!InvAck;		hit; dir!DataAck; → M
M	hit;	hit;	dram(self)!PutX; → MI;		dram(self)!PutX; src!DowngradeAck; → MS	dir!PutX; → I		
MI	stall;	stall;	stall;		src!DowngradeAck;	→ I	→ I	
MS	stall;	stall;	stall;			src!InvAck; → MI	→ S	

Table 2: DRAM Cache Controller

	Replacement	src?GetS	src?GetX	src?Upgrade	src?Inv	src?Data	src?PutX	src?UpgradeAck
I		dir!GetS; → IS	dir!GetX; → IM	dir!Upgrade; → SM_U	llc(self)!Inv;		// forward dir!PutX;	
IS	stall;				llc(self)!Inv; → IS_I	copy data; llc(self)!Data; → S	// forward dir!PutX;	
IS_I	stall;				src!InvAck;	llc(self)!Data; → I		
IM	stall;				src!InvAck;	copy data; llc(self)!Data; → M	// forward dir!PutX;	
S	→ I	src!Data;	dir!GetX; → SM	dir!Upgrade; → SM_U	llc(self)!Inv; → I			
SM	stall;				llc(self)!Inv; → IM	copy data; llc(self)!Data; → M		
SM_U	stall;				llc(self)!Inv; → IM	copy data; llc(self)!Data; → M		llc(self)!UpgradeAck; → M
M	→ I				llc(self)!Inv; → I		copy data; dir!PutX; → S	

Table 3: Directory Controller

	Replacement	src?GetS	src?GetX	src?Upgrade	src?PutX	src?DataAck	src?DowngradeAck	src?InvAck
I		src!Data;	dst ← all − {src}; dram(dst)!Inv; b.so ← {src}; tbe.need_acks ← |dst|; → IM_IA	dst ← all − {src}; dram(dst)!Inv; b.so ← {src}; tbe.need_acks ← |dst|; → IM_IA				

IM_IA	stall;	stall;	stall;	stall;				tbe.need_acks−−;
								if tbe.need_acks = 0 **then**
								dram(b.so)!Data;
								→ IM_DA
								endif

IM_DA	stall;	stall;	stall;	stall;	copy data; llc(src)!PutAck; → MI	→ M		

S	→ I	b.so ← b.so ∪ {src};
		src!Data;
			dst ← b.so − {src};
			dram(dst)!Inv;
			b.so ← {src};
			tbe.need_acks ← |dst|;
			→ SM_IA
				dst ← b.so − {src};
				dram(dst)!Inv;
				b.so ← {src};
				tbe.need_acks ← |dst|;
				if src ∈ dst **then**
				→ SM_U_IA **else**
				→ SM_IA **endif**				

SM_IA	stall;	stall;	stall;	stall;				tbe.need_acks−−;
								if tbe.need_acks = 0 **then**
								dram(b.so)!Data;
								→ SM_DA
								endif

SM_U_IA	stall;	stall;	stall;	stall;				tbe.need_acks−−;
								if tbe.need_acks = 0 **then**
								dram(b.so)!UpgradeAck;
								→ SM_DA
								endif

SM_DA	stall;	stall;	stall;	stall;	copy data; llc(src)!PutAck; → MI	→ M		
M	dram(b.so)!Inv; → MI	llc(b.so)!Downgrade; b.so ← b.so ∪ src; tbe.req ← src; → MS2	dram(b.so)!Inv; b.so ← {src}; → MM_P	dram(b.so)!Inv; b.so ← {src}; → MM_P	copy data; llc(src)!PutAck; → I			

MM_P	stall;	stall;	stall;	stall;	// forward
					dram(b.so)!Data;
					→ MM_DA			

MM_DA	stall;	stall;	stall;	stall;	copy data; llc(src)!PutAck; → MI	→ M		
MS2	stall;	stall;	stall;	stall;	copy data; → MS1		→ MS1	

MS1	stall;	stall;	stall;	stall;	copy data;
					dram(tbe.req)!Data;
					llc(src)!PutAck;
					→ S
							dram(tbe.req)!Data;
							llc(src)!PutAck;
							→ S	

MI	stall;	stall;	stall;	stall;	copy data; → I	→ I		→ I

References

[HKE+16] Cheng-Chieh Huang, Rakesh Kumar, Marco Elver, Boris Grot, and Vijay Nagarajan. C3D: Mitigating the NUMA Bottleneck via Coherent DRAM Caches. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, October 2016.