The Cray BlackWidow: A Highly Scalable Vector Multiprocessor

Dennis Abts dabts@cray.com
Abdulla Bataineh† abb@cray.com
Steve Scott sscott@cray.com
Greg Faanes gjf@cray.com
Jim Schwarzmeier jads@cray.com
Eric Lundberg epl@cray.com
Tim Johnson tjohnson@cray.com
Mike Bye mbye@cray.com
Gerald Schwoerer gfs@cray.com

Cray Inc.
Chippewa Falls, Wisconsin 54729

Abstract

This paper describes the system architecture of the Cray BlackWidow scalable vector multiprocessor. The BlackWidow system is a distributed shared memory (DSM) architecture that is scalable to 32K processors, each with a 4-way dispatch scalar execution unit and an 8-pipe vector unit capable of 20.8 Gflops for 64-bit operations and 41.6 Gflops for 32-bit operations at the prototype operating frequency of 1.3 GHz. Global memory is directly accessible with processor loads and stores and is globally coherent. The system supports thousands of outstanding references to hide remote memory latencies, and provides a rich suite of built-in synchronization primitives. Each BlackWidow node is implemented as a 4-way SMP with up to 128 Gbytes of DDR2 main memory capacity. The system supports common programming models such as MPI and OpenMP, as well as global address space languages such as UPC and CAF. We describe the system architecture and microarchitecture of the processor, memory controller, and router chips. We give preliminary performance results and discuss design tradeoffs.

General terms: design; architecture; performance

Keywords: multiprocessor; shared memory; vector; MPP; fat-tree; high-radix; distributed shared memory

SC07 November 10-16, 2007, Reno, Nevada, USA
ISBN 978-1-59593-764-3/07/0011

†Abdulla Bataineh is also affiliated with Jordan University of Science and Technology, Irbid, Jordan.

1 Introduction

The Cray BlackWidow (BW) vector multiprocessor is designed to run demanding applications with high communication and memory bandwidth requirements. It uses a distributed shared memory (DSM) architecture to provide the programmer with the appearance of a large globally shared memory with direct load/store access. BlackWidow integrates with Cray's XT infrastructure to create a hybrid system, consisting of high bandwidth vector compute nodes, service nodes based on the AMD Opteron processor, and optionally other compute nodes based on AMD Opterons, or FPGAs.

The BlackWidow system consists of four types of chips. The *BW processor* chip includes a 4-way dispatch superscalar core, 8 vector pipes, L1 and L2 caches, and ports to local memory and network. The *Weaver* memory controller chip provides an L3 cache shared across a 4-processor SMP node, memory directory for hardware cache coherence, and DDR2 memory interfaces. The YARC[1] [18] chip is a 64-port high-radix router that is used to implement a folded-Clos network [4]. The *StarGate* FPGA provides a communication protocol bridge between the BW interconenct and the Cray XT [7][8] interconnect.

The Cray XT partition provides a set of*service and input/output* (SIO) nodes, which provide login, programming environment, job scheduling, system administration, global file system and other OS services for the system. The SIO nodes are shared by the BlackWidow compute nodes, optional Cray XT compute nodes, and any other compute nodes in the system, providing a unified user environment. This supports applications with *heterogeneous work flows* that exchange data between the scalar (XT) and vector (BW) nodes through the common file system or via direct memory-to-memory communication.

Unlike conventional microprocessors, each BW processor supports abundant memory level parallelism with up to 4K outstanding global memory references per processor. Latency hiding and efficient synchronization are central to the BW design, and the network must therefore provide high global bandwidth, while also providing low latency for efficient synchronization. The high-radix folded-Clos network [18] allows the system to scale up to 32K processors with a worst-case diameter of seven hops.

[1]YARC stands for 'Yet Another Router Chip', and is also Cray spelled backwards.

The BlackWidow system has a number of innovative attributes, including:

- scalable address translation that allows all of physical memory to be mapped simultaneously,
- load buffers to provide abundant concurrency for global memory references,
- decoupled vector load-store and execution units, allowing dynamic tolerance of memory latency,
- decoupled vector and scalar execution units, allowing run-ahead scalar execution with efficient scalar-vector synchronization primitives,
- vector atomic memory operations (AMOs) with a small cache co-located with each memory bank for efficient read-modify-write operations to main memory,
- a highly banked cache hierarchy with hashing to avoid stride sensitivity,
- a high-bandwidth memory system optimized for good efficiency on small granularity accesses, and
- a cache coherence protocol optimized for migratory sharing and efficient scaling to large system size, combined with a relaxed memory consistency model with release and acquire semantics to exploit concurrency of global memory references.

In this paper, we present the architecture of the Cray BlackWidow multiprocessor. As a starting point, we describe the node organization, packaging and system topology in Section 2. We describe the BW processor microarchitecture in Section 3, the memory system in Section 4, and a number of reliability features in Section 5. Section 6 presents preliminary performance results. Section 7 highlights prior related work. Finally, Section 8 summarizes the key attributes of the BlackWidow system architecture.

2 BlackWidow System Overview

The BlackWidow system is built upon four-processor SMP nodes, interconnected with a high-radix folded-Clos (a.k.a. fat-tree) network. This section describes the node organization, network topology and physical packaging of the system.

2.1 Node Organization

Figure 1 shows a block diagram of a BlackWidow compute *node* consisting of four BW processors, and 16 Weaver chips with their associated DDR2 memory parts co-located on a *memory daughter card* (MDC). The processor to memory channels between each BW chip and Weaver chip use a 4-bit wide 5.0 Gbaud serializer/deserializer (SerDes) for an aggregate channel bandwidth of 16×2.5 Gbytes/s = 40 Gbytes/s per direction — 160 Gbytes/s per direction for each node.

The Weaver chips serve as pin expanders, converting a small number of high-speed differential signals from the BW processors into a large number of single-ended signals that interface to commodity DDR2 memory parts. Each Weaver chip manages four DDR2 memory channels, each with a 40-bit-wide data/ECC path. The 32-bit data path, coupled with the four-deep memory access

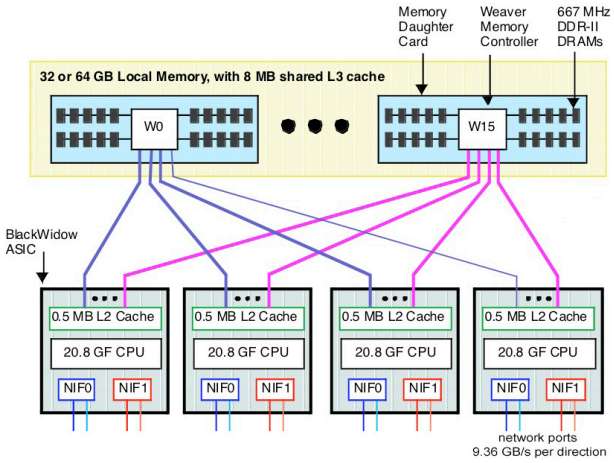


Figure 1. BlackWidow node organization.

bursts of DDR2, provides a minimum transfer granularity of only 16 bytes. Thus the BlackWidow memory daughter card has twice the peak data bandwidth and four times the single-word bandwidth of a standard 72-bit-wide DIMM. Each of the eight MDCs contains 20 or 40 memory parts, providing up to 128 Gbytes of memory capacity per node using 1-Gbit memory parts.

2.2 Network Topology

To reduce the cost and the latency of the network, BlackWidow uses a high-radix, folded-Clos topology, which is modified to permit *sidelinks* amongst multiple peer subtrees. Deterministic routing is performed using a hash function to obliviously balance network traffic while maintaining point-to-point ordering on a cache line basis. A BlackWidow system of up to 1024 processors can be constructed by connecting up to 32 rank 1 (R1) subtrees, each with 32 processors, to rank 2 (R2) routers. A system with up to 4608 processors can be constructed by connecting up to nine 512-processor R2 subtrees via side links. Up to 16K processors may be connected by a rank 3 (R3) network where up to 32 512-processor R2 subtrees are connected by R3 routers. Multiple R3 subtrees can be interconnected using sidelinks to scale up to 32K processors.

The BlackWidow system topology and packaging scheme enables very flexible provisioning of network bandwidth. For instance, by only using a single rank 1 router module, instead of two as shown in Figure 2(a), the port bandwidth of each processor is reduced in half — halving both the cost of the network and its global bandwidth. An additional bandwidth *taper* can be achieved by connecting only a subset of the rank 1 to rank 2 network cables, reducing cabling cost and R2 router cost at the expense of the bandwidth taper as shown by the $\frac{1}{4}$ taper in Figure 2(b).

The YARC chip is a high-radix router[2] used in the network of the Cray BlackWidow multiprocessor. Using YARC routers, each with 64 3-bit wide ports, the BlackWidow scales up to 32K processors using a folded-Clos [4] topology with a worst-case diameter

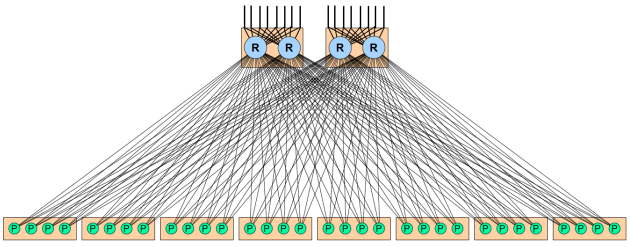
[2]Each YARC port has a peak data rate of 6.25 Gb/s in each direction, however, to tolerate longer network cables, we reduced the target frequency to 5.0 Gb/s

of seven hops. Each YARC router has an aggregate bandwidth of 1.9 Tb/s. YARC uses a hierarchical organization [14] to overcome the quadratic scaling of conventional input-buffered routers. A two-level hierarchy is organized as an 8×8 array of tiles. This organization simplifies arbitration with a minimal loss in performance. The tiled organization also resulted in a modular design that could be implemented in a short period of time.

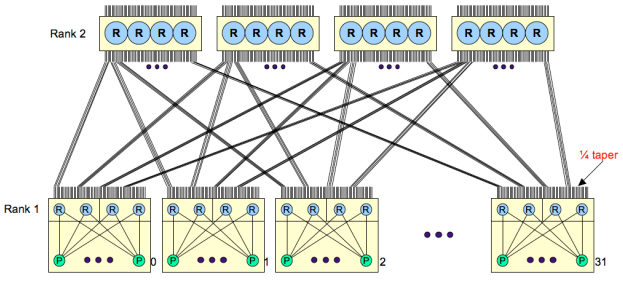
2.3 System Packaging

Each compute module contains two compute nodes, as shown in Figure 3(a) providing a dense packaging solution with eight BW processors and 32 MDCs. At the next level of the hierarchy, as shown in Figure 3(b), a set of eight compute modules and four router cards, each containing two YARC router chips, are connected via a midplane within a *chassis*. The router cards are mounted orthogonally to the compute blades, and each router chip connects to 32 of the 64 processors in the chassis. The chassis contains two rank-1 sub-trees, as shown in Figure 2(a).

All routing within a rank-1 sub-tree is carried via PCB traces within the chassis. All routing between rank-1 sub-trees is carried over cables, which leave the back of the router cards. Two chassis are contained within one compute *cabinet* for a total of 128 BW processors providing an aggregate of ≈2.6 Tflops per cabinet. The BlackWidow system consists of one or more cabinets interconnected with the necessary cables using the high-radix folded-Clos [18] network. Systems above 256 processors include standalone router cabinets containing rank-2 or rank-3 routers.

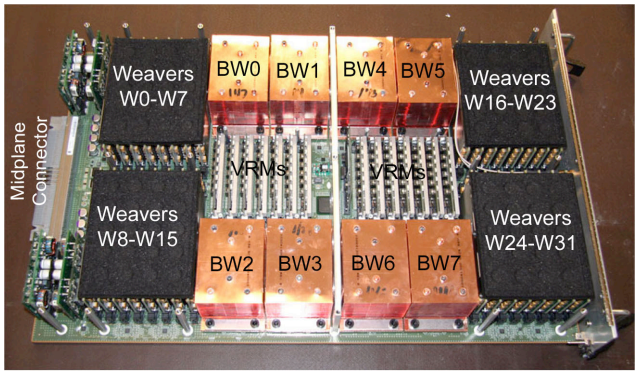


(a) Rank 1 network.

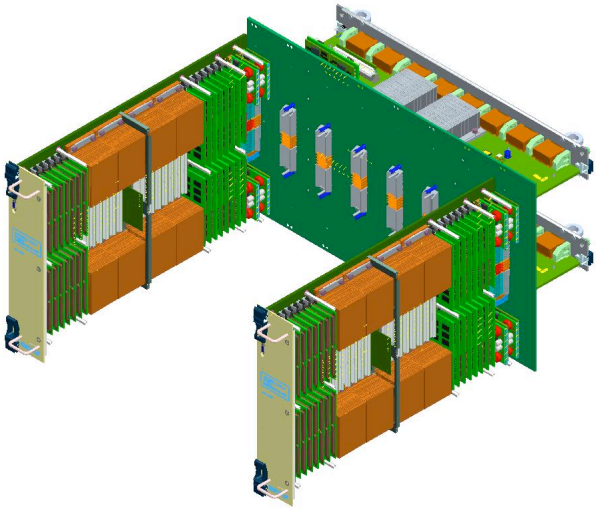


(b) Rank 2 network, shown with a $\frac{1}{4}$ taper.

Figure 2. The BlackWidow high-radix network.



(a) BlackWidow compute module with two nodes.



(b) Chassis with 8 compute modules and 4 network cards.

Figure 3. BlackWidow system packaging.

3 BW Processor Microarchitecture

Unlike the Cray X1 which used an 8-die multi-chip module (MCM) and was implemented in 180nm ASIC technology, each BW processor is a single chip in 90nm technology. Each BW processor is a semi-custom ASIC, where full custom design is used to implement functional unit groups (FUGs) of the 8-pipe vector unit, multi-ported register files, scalar execution units, and L1 cache structures. Standard cells are used for the L2 cache and processor to memory ports. The BW processor is able to achieve higher sustained performance, relative to a convention superscalar design, by exploiting data-level parallelism and hiding memory latency. Latency hiding is achieved with a *decoupled* microarchitecture that exposes more parallelism by (i) decoupling execution of the scalar and vector units enabling *run-ahead* execution of the scalar unit, (ii) decoupling the vector load-store unit (VLSU) from the vector execution unit (VXU), and (iii) allowing vector store instructions to execute in the VLSU before their data has been produced by the VXU, enabling subsequent load instructions to execute while maintaining memory ordering.

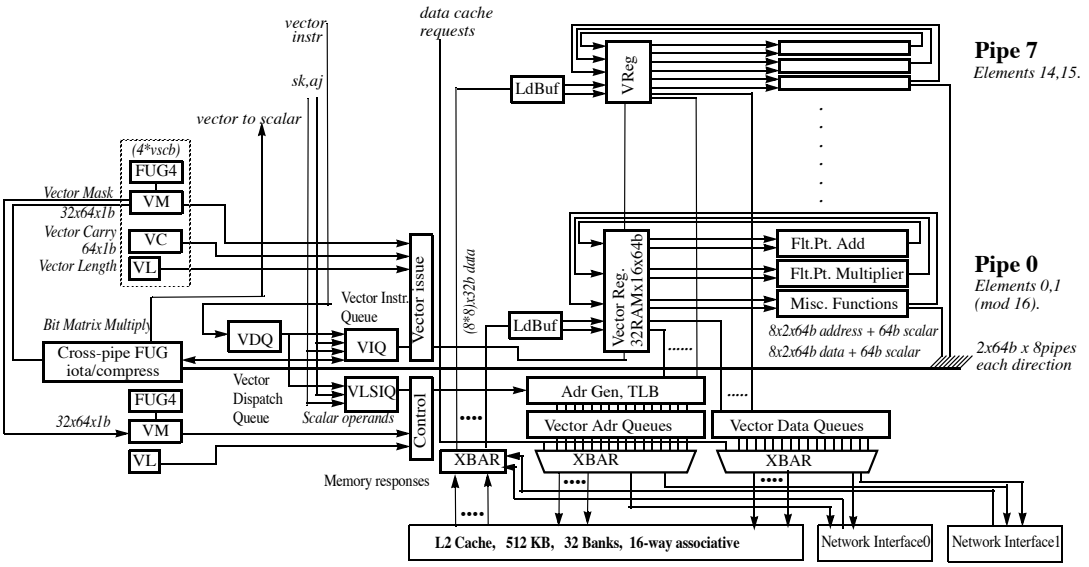


Figure 4. Block diagram of the BlackWidow 8-pipe vector unit, with detail of pipe 0.

3.1 Vector Unit

The BW vector unit is organized as eight pipes[3] (Figure 4) where each pipe is associated with 16 elements of every vector register. The functional units operate at Sclk rate (1.3 GHz in the BlackWidow prototype) while the control for the vector unit executes at Lclk rate (half frequency of Sclk). The BW vector structure has three main units: vector dispatch unit (VDU), vector execute unit (VXU) and vector load/store unit (VLSU). All three vector units are decoupled from the scalar processor and are also decoupled from each other. The vector unit has a total of 24 functional units operating at Sclk rate, a cross-pipe functional unit with a bit matrix multiply (BMM), and a functional unit that operates on vector masks.

The vector unit has a large vector register (VR) file that has 32 physical vector registers with a maximum vector length (MaxVL) of 128 elements. The vector unit also has a vector carry register (VC) that is 1-bit wide and MaxVL long which is used to support extended integer arithmetic. The vector unit has two other register sets that are used to control the vector operations of both the VXU and VLSU. A Vector Length Register (VL) is used to control the length of each vector operation. The vector unit also has 32 vector mask registers (VM) that are 1-bit wide and MaxVL long. The VMs are used to control vector operations on a per element basis.

The VXU is capable of issuing, in order, one vector instruction per Lclk. The VXU can perform both 64-bit and 32-bit vector operations, with the majority of 32-bit operations being performed at twice the rate of 64-bit operations. Vector chaining is supported among the functional units and from functional units to memory. The VLSU can issue, in-order, one load/store instruction per Lclk. It can generate 16 memory references per Lclk for strided loads, gathers, strided stores, or scatters.

The latency to load data from L2 cache or main memory is hidden by decoupling the VXU and the VLSU, thereby allowing the VLSU to run ahead of the VXU. As soon as a vector load/store instruction gets its scalar operands, it is placed in a queue ready to translate in the VLSU. Translated addresses are then immediately sent to the L2 data cache. This decoupling between the VXU and the VLSU is achieved by replicating the VL and VM registers in both units and at the same time allowing instructions that modify the VL or VM registers to execute redundantly in both units.

To support decoupling of the VLSU from the VXU, the vector unit has a *load buffer* (LdBuf) that support up to 4K outstanding 64-bit memory references. The LdBuf is used as a rename buffer for vector load instructions, extending the size of the physical vector register file, until these instructions are ready to write the vector register file. Arithmetic instructions are queued and executed in order once operands become available. Load buffers are significantly more efficient than general-purpose register renaming with respect to their use of physical register space. In a full renaming scheme, physical registers are reserved for all register targets within a loop for each iteration that is unrolled. Load buffers rename *only* the load targets, allowing a given number of physical registers to enable much deeper unrolling.

For stores, translated addresses are sent immediately to the L2 data cache even if the corresponding store data is not available. The VLSU saves steering information for the store data in the *store address buffer* (StAdrBuf). The processor is capable of having up to 4K outstanding vector store addresses before their data has been generated. Store addresses are also sent to the *vector store combining buffer* (VSCB), which is a data structure in front of each bank of the L2 cache. The VSCB allows subsequent vector or scalar loads to bypass earlier vector stores waiting on their data, allowing loop unrolling while preserving memory ordering.

The LdBuf and StAdrBuf facilitate communication between the VXU and VLSU. The size of the LdBuf and StAdrBuf determines the number of instructions that the VLSU is allowed to work ahead of the VXU. To further widen the window of instructions that can be executed in parallel, vector instructions that cannot trap are marked ready to graduate before they even start executing.

[3] Some refer to these as lanes.

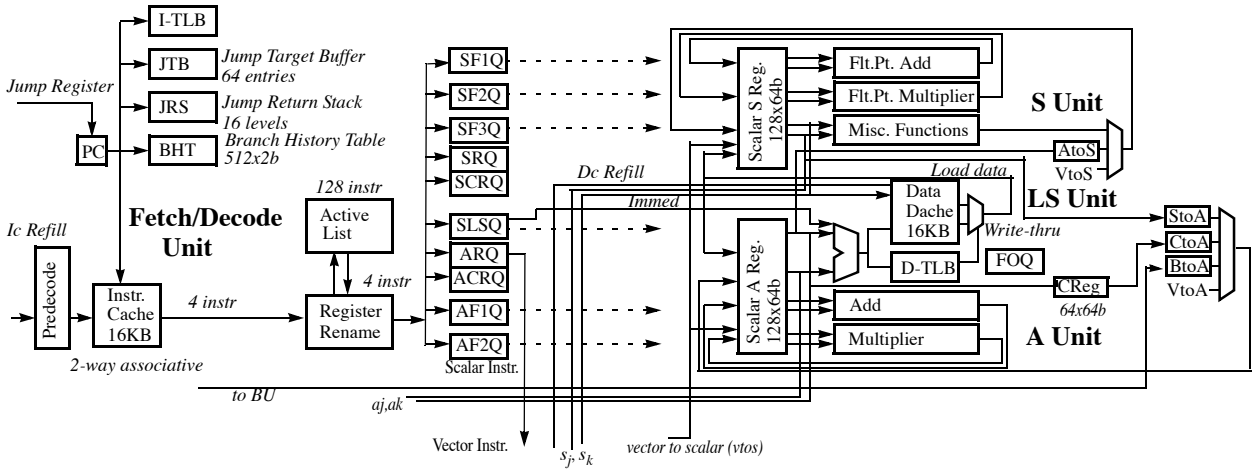


Figure 5. Block diagram of the 4-way superscalar processor.

3.2 Scalar Processor

The BW scalar processor is a 4-way-dispatch superscalar that runs concurrently with the vector unit, so many of the scalar operations are hidden underneath vector execution. It has five main units: Instruction Fetch Unit (IFU), the Dispatch Unit (DU), the AS Unit (ASU), the Scalar Load/Store Unit (SLSU), and the Control Unit (CU). The BW scalar processor dispatches, in-order, up to four instructions per Lclk. It executes instructions out-of-order within the various units and it can graduate in-order up to four instructions per Lclk. The scalar processor also implements speculative execution, register renaming, and branch prediction to allow greater out-of-order execution. It can predict up to four branch or jump instructions and uses a Branch History Table (BHT), a Jump Target Buffer (JTB), and Jump Return Stack (JRS) for improved branch prediction (Figure 5).

The scalar processor has two main 64-bit wide register files, A-Registers (AR) and S-Registers (SR). The ARs are used mainly for address generation. There are 64 logical ARs and 128 physical ARs that are renamed. The SRs are used for both integer and floating-point operations. There are 64 logical SRs and 128 renamed physical SRs.

The BW scalar processor can have 128 speculative instructions in-flight[4], and it is capable of issuing up to four integer instructions that use the A-Registers per Lclk and up to six integer/floating instructions per Lclk that use the S-Registers. Furthermore, the SLSU is completely decoupled from of the rest of the scalar units and it can issue, in-order, one load or store per Lclk. The scalar processor is also able to issue two branch instructions per Lclk, which allows two branch predictions to be resolved per Lclk.

The scalar processor implements two first level caches one for instructions (Icache) and the other one for scalar data (L1 Dcache). The Icache is 16KB, two-way set associative with a cache line size of eight instructions (32 bytes) that are allocated in blocks of four cache lines. The L1 Dcache is write-through 16KB, two-way set associative with a 32-byte cache line size. The Icache is virtually indexed and virtually tagged, while the L1 Dcache is virtually indexed and physically tagged.

[4]Many more non-speculative instructions can be in-flight.

4 BlackWidow Memory Architecture

The BlackWidow system is designed to run distributed memory applications (e.g.: MPI, shmem, UPC, CAF). The machine is globally cache coherent, but allows only memory from the local SMP node to be cached. All remote references are performed as Gets/Puts directly to/from local registers, and are never cached. This provides two benefits. First, it reduces overhead for the expected case of explicit communication between nodes. A Put causes data to flow directly across the network to the target node. If the target line is in the cache, the BlackWidow system will update the cache directly, avoiding any access to main memory by either the Put or subsequent load. A write operation in a ccNUMA machine, on the other hand, causes a line to be fetched *from* the target node and placed dirty into the writer's cache, only to have to be fetched via another remote operation later when the target attempts to access the data. Second, by never caching off-node data, system reliability is significantly improved. If processors are allowed to cache and modify memory from across a machine, then a single failed processor can corrupt memory across the entire machine. A failed processor in the BlackWidow system can corrupt memory only from its local node.

This section describes the address translation mechanism that determines if a memory reference is local or remote and maps it to a physical memory address. It then describes the memory hierarchy accessed by the physical address, and provides a high-level overview of the cache coherence mechanism.

4.1 Address Translation

Virtual memory addresses used for instruction and data reference must be translated from a *virtual* address to a *physical* address before memory is accessed. Figure 6(a) shows the virtual address format. The *memory region* identifies one of three memory regions: *useg* (translated user space), *kseg* (translated kernel space), and *kphys* (untranslated physical memory access by the kernel).

The machine supports two address translation granularities, selectable under OS control. In *node-granularity*, VA[54..42] are treated as a virtual node number, and VA[41..0] are treated as a node memory offset. This supports a hierarchical programming model in which the processors within a node are used to exploit shared memory parallelism underneath a distributed memory model across nodes. In *processor-granularity*, VA[54..40] are treated as a virtual processor number, and VA[39..0] are treated as a processor memory offset. This supports a pure distributed memory model.

Figure 6(b) shows the 57-bit physical memory address, which consists of four fields. The upper 2-bits specify an *address space*, which identifies if the request is destined to main memory or a memory-mapped control register (it is also used by the hardware to indicate a partially translated remote access). All of the memory-mapped registers in the machine are globally accessible via direct loads and stores (though most are accessible only to the OS). Address bits PA[55:42] identify the physical *node* number. Bits PA[41:40] identify either the BW processor for memory-mapped register access, or the quartile of physical memory on the node for main memory references. The remainder of the node's memory offset is given by bits PA[36:0], which allows up to 128 Gbytes of physically addressable memory per processor, or 512 GB per node. Memory pages are spread uniformly across the node, and are accessible to all processors within the node with uniform bandwidth and latency.

BlackWidow supports both *source* and *remote* address translation. The source-based translation uses traditional translation look-aside buffers (TLBs), which map from the virtual address to a physical address in a single step. Each processor contains three independent 136-entry TLBs for translating instruction, scalar, and vector references, respectively. The vector TLB is replicated $\times 8$ for bandwidth.

Figure 7 shows the remote address translation process. The virtual address shown at the top is first checked to determine if it is local or remote, by examining the virtual node/processor field according to the current translation granularity. Local references are translated using source translation as described above. Remote references are translated by first mapping the virtual processor number to a physical processor number using a *node translation table* (NTT) at the source. The NTT provides a maximum of

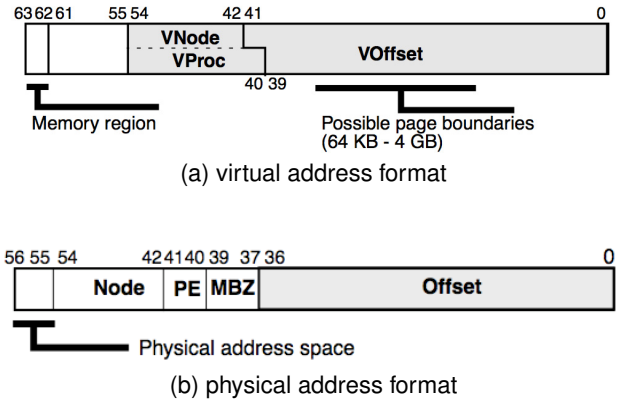


Figure 6. BlackWidow address formats.

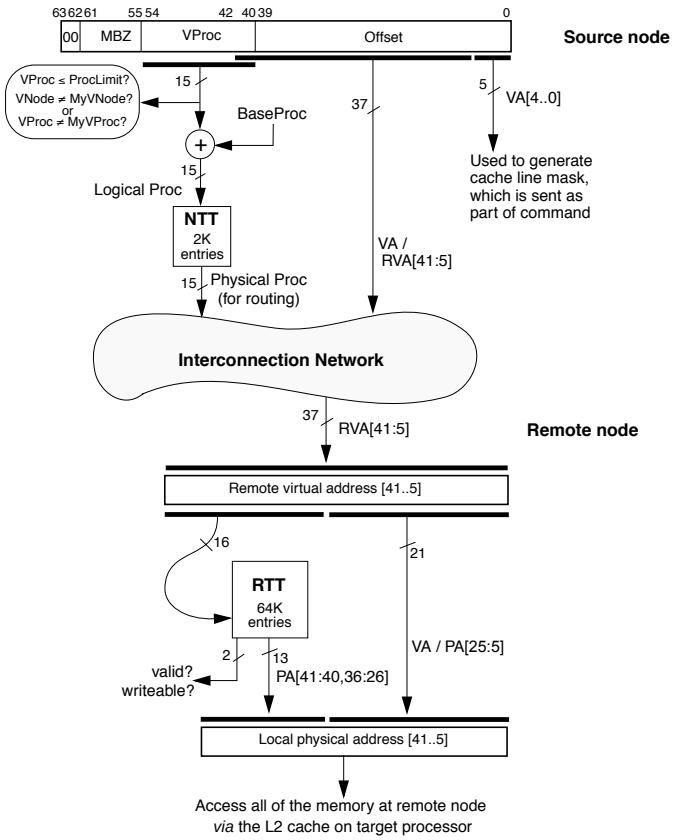


Figure 7. Address translation for remote references.

2K unique translations. For systems with greater than 2K processors, the NTT supports coarser-grain translations of 2, 4, 8, or 16 processors (in which case, the bottom 1, 2, 3, or 4 bits of the virtual processor number are passed straight through translation and the upper address bits are used to index into the NTT).

The remaining remote virtual address (RVA) is then sent to the target processor, where is it is translated to a local physical address using a 64K-entry *remote translation table* (RTT). Thus, each processor needs page translation information about its own node only, which scales efficiently to large numbers of nodes. This mechanism allows the entire system memory to be mapped simultaneously, enabling random accesses to all of global memory with no translation faults.

4.2 Memory Hierarchy

Each BW processor has a 16KB L1 data cache, a 16KB instruction cache, and a 512 KB unified L2 cache. The L1 cache is 2-way associative, and used only for scalar memory references. The L2 cache is 4-way associative and is divided into 32 banks to support high load and store bandwidth to the BW processor. Vector references are sent directly to the L2 cache. To avoid stride sensitivity, the address is hashed before the request is presented to the banks. The L2 cache provides a peak of 166 GB/s of load bandwidth at the prototype processor frequency of 1.3 GHz.

The BlackWidow vector cache was designed for high band-

width (32 independent banks/controllers) rather than a high hit rate (only 512KB). High hit rates are important in traditional scalar machines primarily to reduce effective memory latency. In BlackWidow, however, if even a single vector load element out of 128 misses in the cache, then the load will effectively experience main memory latencies. As a result, the processor is designed to tolerate memory latencies, and the goal of the cache is not to reduce memory latency so much as to reduce the required memory bandwidth. The cache hit rate need only be sufficient to reduce the main memory traffic by the ratio of memory bandwidth to cache bandwidth. The memory concurrency of the processor, however, must be large enough to cover *cache* bandwidth at *memory* latency.

The L3 cache, implemented on the Weaver chips, is 8MB and 16-way associative. The L3 cache is shared by the four BW processors within the local node. Each Weaver chip has four *memory directory* structures and their associated *memory manager* that interfaces to the DDR2 memory channels.

The cache hierarchy is *inclusive*, where the contents of the L2 caches are a strict subset of the L3, and the contents of each L1 Dcache is a subset of the associated L2 cache. Replacement policies are designed to minimize interference with lower levels of the hierarchy. The L2 replacement policy favors retaining lines that are present in the scalar L1 cache. This prevents a vector loop from sweeping through the L2 cache and wiping out the L1 cache. Similarly, the L3 replacement policy favors retaining lines currently held in an L2 cache within a node. The BlackWidow cache line size is 32 bytes, and the minimum access size is 32-bits (single word).

4.3 Coherence

All of the caches in the system are kept coherent via hardware state machines. The L1 scalar data caches are managed as write-through to the L2 caches, and therefore never contain dirty data. This allows vector references to hit out of the L2 without interrogating the L1 caches. The L2 caches are managed as write-back caches. The L3 cache implements a directory-based protocol among the four-processor SMP node.

Each of the 32 L2 cache banks on a processor implements its own coherence protocol engine. All requests are sent through the protocol pipeline, and are either serviced immediately, or shunted into a *replay queue* for later processing. The L2 cache maintains a *state*, *dirty mask* and *replay bit* for every cache line. The L2 also provides a *backmap* structure for maintaining L1 Dcache inclusion information. The backmap is able to track the silent evictions from the L1 Dcache.

The dirty mask is cleared when a line is allocated in the cache, and keeps track of all 32-bit words written since allocation. On a write miss allocation, only words not being written by the vector store are fetched from memory. Stride-1 vector stores are thus able to allocate into the L2 and L3 caches without fetching any data from main memory. On Writeback events, only dirty 8-byte quantities are written back to the L3 cache, and only dirty 16-byte quantities (the main memory access granularity) are written back to DRAM. The *replay bit* indicates a transient line that has one or more matching requests in the *replay queue*. This is used to prevent a later request from passing an earlier replayed request.

Each bank maintains a vector store combining buffer (VSCB) that tracks pending vector write and atomic memory operation (AMO) addresses (including words within the cacheline) that are currently waiting for their vector data. Since the address and data are sent independently to the cache interface, the VSCB is used to combine the store address with its associated data before it is presented to the L2 coherence engine. A load can read part of a line even though another part of the line is waiting for vector data from a previous store. This prevents inter-iteration dependencies due to false sharing of a cache line. The VSCB allows later vector or scalar loads to bypass earlier vector stores waiting for their data. It also provides ordering with respect to other local requests that arrive after a vector write/AMO but before the corresponding vector data.

Each of the Weaver memory controller chips contains four subsections, each containing a directory protocol engine, a piece of the local L3 cache containing 128KB of data, and a memory manager, which handles all memory traffic for the corresponding portion of L3. The directory data structure maintains a simple bit-vector sharing set for each of the L2 lines, and tracks whether the lines are shared, or exclusive to an L2 cache (which may have dirtied the line). The memory directory state is integrated with the L3 cache controller state. A global state in the L3 is used to indicate that data is present in L3 and is consistent with main memory although the data is not cached by any processors. This reduces the number of fill requests to the memory managers for migratory data sharing patterns.

Cache data arrays are protected using a single-bit error correction and double-bit error detection (SECDED) code. If a corrupted line is evicted from the cache, the data is *poisoned* with a known bad ECC code when the cache data is written back to main memory. Evicting a corrupted cache line will not cause an immediate exception, instead the exception is deferred until the corrupted data is consumed.

Three virtual channels (VCs) are necessary to avoid deadlock and provide three-legged cache transactions within the SMP node. Requests and responses are segregated on VC0 and VC1, respectively. Cache interventions are sent on VC1, and intervention replies travel on VC2. The memory directory is guaranteed to sink any incoming VC2 packet, and is allowed to block incoming requests on VC0 (i.e. no *retry nacks* are used) while waiting to resolve transient states. Blocked requests are temporarily shunted into replay queues, allowing other, unrelated requests to be serviced. The instruction cache is kept coherent via explicit flushing by software when there is a possibility that the contents are stale.

4.4 Memory Consistency

The BlackWidow system implements the NV-2 instruction set architecture, which precisely defines the memory ordering guarantees that programs can assume. Individual load/store instructions can provide cache hints, such as “no allocate,” to the hardware. The hints allow the hardware to optimize caching behavior for performance, but do not change the functional behavior of the instruction.

4.4.1 Single processor ordering

The hardware provides minimal guarantees on the ordering of the effects of vector memory reference operations with respect to each

other and to scalar memory references. Explicit memory synchronization instructions must be used to guarantee memory ordering between multiple vector memory references and between vector and scalar memory references. The *program order* is a total ordering relation between operations on a single processor as a sequence observed by a simple fetch/execute cycle. A *dependence order* is a partial ordering relation, a subset of program order, where two operations x and y are ordered with respect to dependence if x is earlier than y in program order, and at least one of the following conditions hold:

- operations x and y are scalar references or atomic memory operations (AMOs) to the same address,
- a synchronization instruction executes between memory references x and y in program order that orders x and y,
- y has a *true* dependence on x,
- x and y are elements of the same ordered scatter instruction,
- x and y are elements of a vector store with as stride value of zero, or
- x and y are elements of the same vector atomic memory operation (VAMO).

Operations that are ordered by dependence will appear to each processor issuing them to execute in program order. They will not necessarily appear to execute in the same order by *other* processors in the system. A suite of `Lsync` instructions are available to provide additional intra-processor ordering when needed.

4.4.2 Multiple processor ordering

There are three ordering rules that apply only to memory references performed and observed by multiple processors: (i) writes to the same 32-bit (single word) are *serialized* – that is, for any given program execution, writes to a given location occur in some sequential order and no processor can observe any other order, (ii) a write is considered *globally visible* when no processor can read the value produced by an earlier write in the serialized order of writes to that location, and (iii) no processor can read a value written by *another processor* before that value becomes globally visible. When ordering between multiple processors is required, a `Gsync` instruction is used to ensure that all earlier writes are visible and all earlier reads are bound to a value. For more efficient lock manipulation, *release* (`Gsync R`) and *acquire* (`Gsync A`) instruction variants provide only the necessary ordering guarantees to support inter-processor synchronization for locks and critical sections.

5 Robust System Design

The BlackWidow memory system is highly parallel, with 64 independent memory controllers for each four-way node, providing a maximum memory capacity of 128 Gbytes of DDR2 memory with up to 640 2-Gbit DDR2 devices per node. With an estimated SER of 10 FIT per DDR2 device, we are faced with the daunting task of building reliable *systems* [15] from increasingly unreliable *components* [13].

Large memory structures and inter-chip communication paths are all protected with SECDED and CRC codes. Furthermore, each memory reference is tagged with a k bit which indicates whether the reference was generated by a user-level application (k=0), or the operating system (k=1). This information allows the system software handling the exception to easily differentiate errors occurring at the user-level (which would result in an application failure) from those taken at the system-level (which could result in a system crash).

The fault-tolerant DRAM scheduling, and hardware support for scrubbing, auto-retry, and spare-bit insertion address the growing problem of soft errors in DRAM devices. Many of these techniques work transparently together to improve both *application* and *system* reliability.

5.1 Memory Scrubbing

Assuming that soft errors follow a uniform distribution in memory, the longer a word of data lives in DRAM, the more likely it will be exposed to the effects of any number of soft errors. In the worst case, a sufficient number of bits will be upset resulting in silent data corruption. The Weaver memory manager implements a *hardware-based memory scrubbing* engine capable of cycling through memory, reading and correcting any single-bit errors by writing back corrected data. In order to make the scrubbing engine as non-intrusive as possible, it is desirable to perform scrub reads when the DRAM channel is idle. At the same time, certain quality of service (QoS) guarantees must be made, ensuring that the entire DRAM part is scrubbed with a specifiable frequency. To satisfy these requirements, the Weaver memory manager uses a scheme in which a DRAM scrub cycle is broken up into fixed periods, each of which includes a single scrub read request. Each scrub period is divided into two distinct time regions, the first will perform an early scrub read if no other traffic is present at the eight-to-one bank request arbiter. However, at some point the scrub request must be considered a priority, and in the second phase of each period, user requests will be blocked to make way for the pending scrub request.

5.2 Memory Request Auto-Retry

In addition to conventional ECC, Weaver implements a request *auto-retry* mechanism for read operations that experience non-poisoned multi-bit errors (MBE). The retry protocol allows multiple attempts to read the memory and disambiguate a persistent MBE soft error that caused multiple "bit flips" in the memory device from a soft error induced by electrical noise, operating temperature variance, or marginal electrical signaling. Furthermore, it is possible for a combination of errors to exist simultaneously and give the appearance of a single event causing multiple-bit errors. For example, a single bit flip event in main memory, combined with electrical noise on the memory interface pins can result in a soft error that contains multiple bits in error. By *retrying* a faulty memory read operation, the hardware can distinguish between an intermittent error and a persistent error.

All memory references that return a non-poisoned error response are retried. When an MBE is detected, the memory sequencer will immediately (subject to the bank cycle time of the device) schedule the retry operation. Arbitration logic in the memory sequencer gives the retry request priority so that no other requests are allowed to be reordered in front of the retry operation; that is, the next reference to the bank where the MBE occurred is

guaranteed to be the retry. This retry operation is attempted and, if the retry fails, the memory sequencer returns an error response and the error is logged as a persistent MBE in the error table. An exception is raised to the operating system so the application that observes the error can be terminated, and the memory page with the faulty location can be removed from the free-page list.

5.3 Spare-bit Insertion

The five ×8 DDR2 devices in a memory channel provide 40 bits worth of storage, with 32 bits for data and 7 bits for ECC. The remaining unused bit can be multiplexed into the data path using a series of two-to-one multiplexers to replace faulty bits in memory. The control of each mux is selected individually according to the bit position that is to be skipped so that the "spare" bit is used instead. The spare-bit logic is an adjunct to the refresh/scrubbing logic. If a spare-bit insertion sequence is active, the scrub logic will perform a 32-byte read-modify-write sequence to insert the new spare-bit selection. Since, the spare-bit cannot be inserted instantaneously, there is a mechanism to track which memory words use the newly inserted spare-bit location and which use the previous bit position. A single address pointer is used to mark the boundary between locations that use the *new* bit position and those locations with the old spare-bit locations.

In order to determine which bit positions make for good spare-bit candidates, a set of single-bit error *histograms* are provided – with a bin for each of the 40 bit positions. These histograms track the location of all single bit errors and can be periodically polled by system software. Commonly observed single-bit errors will be exposed by reading the histogram registers and examining the frequency of errors. Using the histogram data provides an informed decision about which failing bit to replace with spare-bit insertion.

6 Performance Results and Discussion

This section describes our preliminary results on prototype hardware and discusses design trade-offs. Our early performance evaluation was performed on a system containing 11 compute blades (22 compute nodes) organized as a rank 1.5 fat-tree network. The BW processor clock frequency was running at the prototype frequency of 1.3 GHz, the processor-to-memory SerDes were running at 5.0 GHz, and network SerDes links were operating at 2.5 Gbps (50% of the target frequency for a production system).

We compared the BlackWidow system against an HP XC300 system using Intel Woodcrest processors. Table 1 summarizes the system configurations that are being compared. We show results for the HPCC [11] benchmark suite in Table 2 and also the aerodynamic computational fluid dynamics (CFD) code `OVERFLOW` in Table 3.

The BlackWidow L2 cache delivers 166 Gbytes/s of load bandwidth for each BW processor at 1.3 GHz, for a ratio of 8 bytes per flop. The main memory system delivers 122 Gbytes/s load bandwidth for the four-processor node or ≈1.5 bytes per flop. This high bandwidth is necessary to sustain high fraction of peak on real applications.

Table 2 shows a comparison for a subset of HPCC Benchmarks for three processor (socket) counts using the results from an HP

Table 1. System configurations for benchmarking.

System Name	XC3000	BlackWidow
Manufacturer	HP	Cray
Chip	Intel Woodcrest	Cray BlackWidow
Processor type	Core 2 Duo	BW
Processor speed	3.0 GHz	1.3 GHz
Socket peak	24 Gflops	20.8 Gflops
Processor count	32-128	32-88
Threads	1	1
Interconnect	InfiniBand 4× 5.0 Gb/s (DDR)	YARC 12× 2.5 Gb/s
MPI	HP MPI 02.02.05.00	Cray MPICH2 1.0.4

XC3000 [5] system with Intel Woodcrest (Core 2 duo) processors [12] and the Cray BlackWidow system. The WoodCrest socket has two cores and the Blackwidow socket only one core, however, we are comparing only socket-to-socket. For EP-DGEMM, WC achieves about 78% of peak and BW about 88% of peak. For PTRANS the BlackWidow system shows a 4.8× advantage over the Woodcrest system, even though BlackWidow is operating only 50% of target network bandwidth (which is less than $\frac{1}{2}$ the signaling rate of the Woodcrest system in this comparison). At 64 sockets the BlackWidow system has a 3.8× bandwidth advantage over the Woodcrest system for PTRANS, and 4.6× for Random Ring. We expect both PTRANS and Random Ring bandwidth performance to increase substantially at target network bandwidth for the BlackWidow system.

On BlackWidow, the update loop of the G-RA (Global RandomAccess) benchmark is concisely implemented as a PGAS loop using either CAF (See Figure8) or UPC. Because of vector atomic updates and global addressability of the BlackWidow system, the 64-core CAF version of G-RA shows an *enormous* advantage over the Woodcrest system — a factor of 135×! On the BlackWidow system the CAF version of G-RA is totally dominated by network bandwidth, so we expect the factor of 135 to grow to 270× at target network SerDes rate. With a full fat-tree network, rather than a rank 1.5 network, the BlackWidow system would show even higher GUPS performance.

For EP-STREAMS the BlackWidow system shows a 16× advantage over the Woodcrest system. This is a reflection of BWs ability to deliver high memory bandwidth when an application re-

```
*** GUP (CAF for multipe PEs)

integer, parameter :: ransize=512

do i=1, pe_updates/ransize
!dir$ concurrent
  do j=1,ransize
    if ( ran(j) < 0) then
      q = 7
    else
      q = 0
    endif
    ran(j) = shiftl(ran(j),1) .xor. q
    idx =shiftr(iand(ran(j),ixmask)*nimages,l2pes)
    ipe =shiftr(idx,l2pe_tabsz)
    ioff =iand(idx, pe_tabsz-1)
    table(ioff+1)[ipe+1]=table(ioff+1)[ipe+1] .xor. two
  enddo
enddo
```

Figure 8. Co-array Fortran (CAF) implementation of G-RA (RandomAccess) benchmark.

Table 2. HPCC benchmarks for an HP XC3000 system with Intel Woodcrest (WC) and a BlackWidow (BW) prototype system.

System	Number of sockets	G-PTRANS (Gbytes/s)	G-RA[a] (GUPS)	EP-STREAM (Gbytes/s)		EP-DGEMM (GFlops)	RandomRing (Gbytes/s)	(μs)
				system	triad		bandwidth	latency
HP XC3000 (Woodcrest)	16	5.142	0.0415	49.15	1.536	9.29	0.300	8.87
	32	8.978	0.0446	98.56	1.540	9.49	0.272	8.99
	64	17.62	0.0402	197.6	1.544	9.26	0.242	9.17
BlackWidow	16	17.06	1.47	399.8	24.99	18.70	1.39	9.52
	32	24.62	2.75	752.3	23.51	18.36	1.20	10.43
	64	67.03	5.41	1619	25.31	17.98	1.12	11.57

[a]BlackWidow results use Co-Array Fortran (CAF) (Figure 8) implementation.

quires it. Similarly, even at 50% network bandwidth and rank 1.5 network, the BlackWidow system delivers 4.8× higher Random Ring bandwidth than the Woodcrest system. The BlackWidow system is designed to achieve very high network bandwidth for demanding operations like global transposing. Random ring latency is about 21% faster on the Woodcrest system than the BW system. However, with the option of CAF or UPC on the BlackWidow system a user can reduce network latency considerably.

Table 3 shows a comparison between the BlackWidow and Clovertown systems for two standard benchmarks of the `OVERFLOW` aerodynamics code. For the purposes of this paper the benchmarks are labeled as "test1" and "test2". Each test case has the same number of grids (7) and the same number of points (1.119 million). Test1 runs 100 time steps on a coarse grid, then 100 time steps on a medium grid and then 20 time steps on a fine grid. Test2 runs 100 time steps on a coarse grid, 100 time steps on a medium grid and then 400 time steps on the fine grid. Since test2 runs longer on the fine grid, test2 sees a longer effective vector length over the course of the run. `OVERFLOW` generally requires high memory bandwidth, so we choose that code to illustrate the BlackWidow systems ability to deliver high memory bandwidth on a real application. In terms of characterizing test1 compared to test2, test2 has finer grids, which translates to longer vector length operations on BlackWidow, compared to test1.

As can be seen from the results of Table 3, the BW processor holds a 4× to 6× advantage over a single core of the Clovertown system. When four MPI processes are running, the BlackWidow holds a 4.8× to 8.9× advantage over the Clovertown system. These results illustrate that BW is designed to deliver high memory bandwidth on applications like `OVERFLOW`, for which time-to-solution is important.

Table 3. `OVERFLOW` benchmark comparing BlackWidow and Intel Woodcrest processors.

			`Overflow 2.0aa`	
Processor (GHz)	Num of Sockets	Num of MPI processes	test1 (sec)	test2 (sec)
Clovertown[b] 2.33 GHz	1	1	310	3634
	2	4	168	1977
BlackWidow 1.3 GHz	1	1	71	574
	4	4	35	222

[b]Using the Intel compiler on a dual-socket Xeon 5345 motherboard, each operating at 2.33 GHz with 2×4 MB cache, 1.3 GHz front-side bus, and 8×2Gbyte of FBDIMM 667 ECC/REG main memory.

The benchmark results of Tables 2 and 3 give evidence that the BlackWidow system is designed to deliver very high memory and network bandwidth for demanding applications. Other special features like vector atomic updates, global addressability, and a state-of-the-art vectorizing compiler enable uniquely high performance on demanding algorithms. Codes that run well from cache, on the other hand, or that do not vectorize well, are better suited for commodity microprocessors. The BlackWidow nodes are therefore well suited for inclusion in a hybrid system.

7 Related Work

The predecessor to the Cray BlackWidow was the Cray X1 [6][1] multiprocessor which introduced a new instruction set architecture (NV-1) to the traditional Cray vector machines. The Cray X1 provided a multi-chip module (MCM) processor with a total of 12.8 Gflops. The X1 was the first globally cache-coherent [19] distributed shared memory vector machine [20][3]. The X1 was the first machine to implement a *decoupled* vector microarchitecture. The NV-1 ISA introduced new synchronization primitives to allow decoupling between the vector and scalar execution units, and decoupling between the vector load-store unit (VLSU) and the vector execution unit (VXU). Run-ahead execution and vector-scalar decoupling [10][21] and subsequent work [2] showed the benefits of a decoupled vector microarchitecture. The Cray X1 showed significant performance benefits [22] for vector codes, however, it was less cost-competitive for codes that did not exhibit a high degree of parallelism.The authors of [17] compare several parallel vector architectures, and show the advantages that parallel vector machines bring to bear on four scientific applications: FVCAM, LBMHD3D, PARATEC, and GTC.

The Tarantula [9] machine extended the Alpha EV8 microarchitecure and ISA to directly support vector operations of up to 128 64-bit elements. This included modifications to the EV8 cache coherence protocol and attached directly to the EV8's L2 cache interface. They showed that with an aggressive superscalar execution unit and conventional cache hierarchy, combined with an efficient vector engine (Vbox), they could achieve significant speedup — up to 5× speedup, relative to the baseline EV8 design, on floating-point intensive scientific workloads. Furthermore, they showed that vector extensions to the EV8 provided significant "real computation" per transistor and per watt ratios, with Tarantula providing 0.55 Gflops/watt compared to the CMP-EV8 baseline design of 0.16 Gflops/watt.

The NEC SX-8 [16] operates at 2 GHz, twice the frequency of the SX-6, and contains 4 vector pipes to produce a peak of 16 Gflops per processor. The SX-8 has an 8-processor node which are then clustered together via a custom IXS network. Similar to the BlackWidow, it uses commodity DDR2 memory parts. The SX-8 main memory provides 41 Gbytes/s of peak STREAM bandwidth or 2.56 bytes/flop, approximately 2× that of the BlackWidow. The SX-8 is designed to provide high memory bandwidth within an SMP node, and does not provide global addressability across the machine. Inter-node bandwidth is modest by comparison, and the the machine scales to at most 512 nodes (4K processors).

8 Conclusion

The performance of a parallel computer system is often limited by the memory and network latency and bandwidth characteristics. The BlackWidow system is designed for applications with high memory and network bandwidth requirements. In this paper we describe the BlackWidow system architecture in terms of the system topology, packaging, processor, memory and network architecture. The BlackWidow system scales to 32K processors, providing floating-point intensive applications with 20.8 Gflops per processor operating at the prototype frequency of 1.3 GHz, and 122 GB/s of local memory load bandwidth within a 4-processor SMP node. The cache and memory architecture are highly banked to support a high degree of memory concurrency and efficiently support irregular access patterns (non-unit strides and gather/scatter).

We describe BW processor microarchitecture which allows decoupling between the vector load/store unit and vector execution units. The BW superscalar processor runs concurrently with the vector unit allowing many of the scalar operations to be hidden underneath vector execution. Unlike a multi-core superscalar processor, which can only tolerate a handful of outstanding cache misses, the load buffer in the BW vector unit supports up to 4K outstanding global memory references.

A scalable address translation mechanism allows all the system memory to be mapped simultaneously. All the caches in the system are kept coherent using hardware coherence engines in the L2 and L3 caches. The L3 cache implements a directory-based protocol among the four-processor SMP node, but all remote references are performed as Get/Put operations directly to/from local registers. By not caching remote memory references, we reduce the overhead of explicit communication between nodes. The relaxed memory consistency model allows abundant parallelism in the memory system, and the NV-2 ISA defines primitives for efficient synchronization of vector-scalar execution and multi-processor ordering.

The BlackWidow system is designed for high single-processor performance, especially for codes with demanding memory or communication requirements. Simple packaging allows much lower cost per processor than previous Cray vector systems, and the architecture is designed to scale efficiently to large processor counts. BlackWidow interfaces with a Cray XT system to create a hybrid system with a unified user environment and file system. In this context, the BlackWidow processor provides accelerated performance on the subset of codes that are well suited for vectorization and/or a more aggressive memory system and network.

Acknowledgments

We would like to thank the entire BlackWidow development, applications, and benchmarking teams.

References

[1] D. Abts, S. Scott, and D. J. Lilja. So many states, so little time: Verifying memory coherence in the Cray X1. In *IPDPS 03, In the Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE Computer Society, April 2003.

[2] C. Batten, R. Krashinsky, S. Gerding, and K. Asanovic. Cache refill/access decoupling for vector machines. In *MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pages 331–342, Washington, DC, USA, 2004. IEEE Computer Society.

[3] C. Bell, W.-Y. Chen, D. Bonachea, and K. Yelick. Evaluating support for global address space languages on the Cray X1. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 184–195, New York, NY, USA, 2004. ACM Press.

[4] C. Clos. A Study of Non-Blocking Switching Networks. *The Bell System technical Journal*, 32(2):406–424, March 1953.

[5] Condensed results for HPCC Challenge Benchmarks. http://icl.cs.utk.edu/hpcc/hpcc_results.cgi.

[6] Cray X1. http://www.cray.com/products/x1/.

[7] Cray XT3. http://www.cray.com/products/xt3/.

[8] Cray XT4. http://www.cray.com/products/xt4/.

[9] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec. Tarantula: a vector extension to the alpha architecture. In *ISCA '02: Proceedings of the 29th annual international symposium on Computer architecture*, pages 281–292, Washington, DC, USA, 2002. IEEE Computer Society.

[10] R. Espasa and M. Valero. A simulation study of decoupled vector architectures. *Journal of Supercomputing*, 14(2):124–152, 1999.

[11] HPCC Challenge Benchmarks. http://icl.cs.utk.edu/hpcc/.

[12] Intel Core2 Duo. http://www.cray.com/products/xd1/.

[13] A. Johnston. Scaling and Technology Issues for Soft Error Rates. In *Proceedings of the 4th Annual Research Conference on Reliability*, Stanford, CA, October 2000.

[14] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 420–431, Madison, WI, USA, 2005. IEEE Computer Society.

[15] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. The Soft Error Problem: An Architectural Perspective. In *Proceedings of the 11th International Conference on High-Performance Computer Architecture (HPCA2005)*, 2005.

[16] NEC SX-8 Vector supercomputer. http://www.nec.co.jp/press/en/0410/2001.html.

[17] L. Oliker, J. Carter, M. Wehner, A. Canning, S. Ethier, A. Mirin, D. Parks, P. Worley, S. Kitawaki, and Y. Tsuda. Leading computational methods on scalar and vector hec platforms. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 62, Washington, DC, USA, 2005. IEEE Computer Society.

[18] S. Scott, D. Abts, J. Kim, and W. J. Dally. The BlackWidow High-radix Clos Network. In *ISCA '06: Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pages 16–28, Boston, MA, June 2006.

[19] S. Scott and A. Bataineh. U.S. Patent: Optimized high-bandwidth cache coherence mechanism, http://www.patentstorm.us/patents/7082500.html. 2006.

[20] H. Shan and E. Strohmaier. Performance characteristics of the Cray X1 and their implications for application performance tuning. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 175–183, New York, NY, USA, 2004. ACM Press.

[21] J. E. Smith. Decoupled access/execute computer architectures. In *ISCA '82: Proceedings of the 9th annual symposium on Computer Architecture*, pages 112–119, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.

[22] J. T. H. Dunigan, M. R. Fahey, J. B. W. III, and P. H. Worley. Early evaluation of the Cray X1. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 18, Washington, DC, USA, 2003. IEEE Computer Society.