The Cray Programming Environment for BlackWidow

Luiz DeRose, Terry Greyzck, Mary Beth Hribar, Brian Johnson, Mark Pagel, Howard Pritchard, and Greg Titus

Cray Inc

ABSTRACT: *The BlackWidow Programming Environment provides a natural follow on to the Cray X1/X1e software. It focuses on optimization and easy of use. In this paper we describe the main features and enhancements of the Cray BlackWidow programming environment, which is being designed to help users achieve the highest possible performance from the hardware.*

KEYWORDS: BlackWidow, Programming environment, programming models, compilers, scientific libraries, application tools.

1. Introduction

In this paper we describe the Programming Environment for the Cray BlackWidow vector system, which is a follow on to the Cray X1E system. The BlackWidow programming environment is being designed to help users achieve the highest possible performance from the hardware. In addition, by continuing the move towards a common software infrastructure among Cray products, it will provide a natural follow on to the Cray X1E systems.

2. Overview of the BlackWidow System

In this section we describe some of the hardware and software features that are significant to the programming environment.

The Cray X1 and X1E systems require automatic multi-level parallelism, at the shared memory processor level (multistreaming) and at the vector level (vectorization) for full utilization of the machine. With the switch to harnessing the power of a single processor, the need for multistreaming has been eliminated from the BlackWidow architecture. This allows the compiler to focus on the much simpler task of automatic single level parallelism, at the vector level.

Similarly to the Cray X1, the Cray BlackWidow system uses a remote translation table to link the address spaces of several nodes into one, which allows the referencing of off node memory with simple loads and stores. This allows users to achieve higher parallel performance with the use of PGAS languages and the use of one-sided communication mechanisms like Cray SHMEM.

While the Cray X1E used a big-endian data representation; the Cray BlackWidow system switches to use a little-endian representation. The data sizes and layouts are as specified in the AMD64 ABI, with the exception that 80-bit floating point is not supported (128-bit floating point is used instead). For most user codes, this endian change is transparent.

3. Parallel Programming Models

The programming environment of the Cray BlackWidow system supports an extensive set of programming models, including distributed memory approaches like MPI and SHMEM, shared memory mechanisms like OpenMP, and the PGAS languages UPC and Co-Array Fortran. In this section we describe the main characteristics of each of these models.

3.1 MPI

The Cray BlackWidow MPI is based on the MPICH2 from Argonne National Laboratory (ANL), with a custom ADI3 device to better exploit BlackWidow's distributed, shared memory architecture. The ADI3 device uses a virtualization method to allow for multiple methods for message delivery. For example, a special very low latency method is used for message notification between processes on the same node, while a more scalable, but somewhat higher latency method is used for inter-node message notification. This virtualization approach allows for reuse of the Cray BlackWidow ADI3 device on future Cray platforms. The design also facilitates reuse of this core technology within other MPI implementations, should that prove to be desirable.

In addition to point-to-point optimizations within the ADI3 device, collective optimizations employed in the X1 MPI implementation will be adapted for use in the MPICH2 framework. The vector AMO capabilities of the Cray BlackWidow processor and the network topology will allow for additional enhancements to the algorithms employed in the Cray X1 MPI implementation. The initial set of optimized collectives will include:

- MPI_Barrier,
- MPI_Win_fence,
- MPI_Allreduce,
- MPI_Reduce,
- MPI_Bcast,
- MPI_Gather(v),
- MPI_Scatter(v),
- MPI_Allgather, and
- MPI_Alltoall(v,w).

All features of MPI-2 remote memory access (RMA) will be supported in the first Cray BlackWidow MPT release, including use of derived types in MPI_Get, MPI_Put, and MPI_Accumulate operations. Passive RMA operations will also be supported. RMA is optimized for Cray BlackWidow's distributed, shared memory architecture.

Additional MPI-2 features such as dynamic process management support (MPI_Comm_spawn) and thread safety will be made available in subsequent releases of the Cray BlackWidow MPI software.

3.2 Cray SHMEM

The logically shared, distributed memory access (Cray SHMEM) routines provide low-latency, high-bandwidth communication for use in highly parallelized scalable programs and will be fully supported on the Cray BlackWidow architecture.

The implementation builds off of the highly optimized X1E version and will include further optimizations of the SHMEM barrier routine for the first release. Additional optimizations will be done in subsequent releases.

3.3 OpenMP

OpenMP provides multithreaded, shared-memory parallelism using directives to specify data locality, work distribution, and control flow. The API is available for both C/C++ and Fortran. OpenMP for Cray BlackWidow will be equivalent to that for X1, and will conform to the OpenMP 2.0 API. OpenMP concurrency will span at most one node, with a maximum thread count of 4.

3.4 Co-Array Fortran and UPC

Co-Array Fortran (CAF) and Unified Parallel C (UPC) are dialects extending Fortran and C, respectively, to allow programmers to specify both data distribution and work distribution in a single program multiple data (SPMD) programming model. The CAF and UPC support in the compiler is closely integrated with the optimizer, allowing the emitted code to take full advantage of the Cray BlackWidow memory architecture for low latency communication.

CAF and UPC for the Cray BlackWidow system will be equivalent to that for the Cray X1 system. CAF will initially conform to the original 1998 CAF specification plus extensions. However, since, CAF will become an official part of the Fortran language in the next standard, the Cray BlackWidow CAF will evolve to conform with the Fortran standard as that is codified. UPC will conform to the UPC 1.2 specification.

4. BlackWidow Programming Languages

C, C++ and Fortran are the languages supported by the Cray Programming Environment compilers on the BlackWidow system. They are based on the X1 compilers. These compilers will carry forward most of the user interface and functionality of the Cray X1 compilers, including command-line options, pragmas, directives, listing, loopmark, and inlining capabilities. In addition, Cray will provide a gcc compiler that targets the BlackWidow architecture.

4.1 C Language Support

The Cray BlackWidow C compiler supports ISO/IEC 9899:1999 commonly called C99. There are some minor, and little used features that are not fully supported. The Cray C compiler makes use of front-end technology from EDG and supports some common gcc language extensions.

4.2 C++ Language Support

The Cray BlackWidow C++ compiler supports ISO/IEC 14882:1998 with some exceptions. It is based on the same EDG front-end technology. Cray plans to use the gnu g++ runtime library for BlackWidow.

4.3 Gcc for BlackWidow

Cray plans to release a gcc compiler that targets the BlackWidow architecture. It is intended for command builds or for codes that depend on complete gcc compliance. Cray will support dynamic linking of commands with this compiler.

4.4 Fortran Language Support and Fortran 2003 Status

The Cray BlackWidow Fortran compiler is based on the X1 compiler front end and is intended to be fully Fortran 2003 compliant, ISO/IEC 1539:1-2004 by 2007. Good progress has been made toward that goal and a growing set of F2003 features are now available in our Cray X1 compiler.

As of the Cray programming environment release 5.5 the following F2003 features are supported:

- Basic syntax enhancements
- PROTECTED attribute
- VOLATILE attribute
- INTENT attribute for pointers
- mixed PUBLIC and PRIVATE component attributes
- allocatable components
- allocatable dummy arguments
- allocatable function results
- allocatable array assignment
- ASSOCIATE construct
- intrinsic modules
- ISO_FORTRAN_ENV module
- ISO_C_BINDING module
- C interoperability
- IMPORT statement
- PROCEDURE statement
- procedure declaration and abstract interfaces
- procedure pointers
- pointer assignment lower bounds
- pointer rank remapping
- FLUSH statement
- IOMSG keyword in I/O statements
- MAX and MIN with character arguments
- NEW_LINE intrinsic
- GET_COMMAND intrinsic
- COMMAND_ARGUMENT_COUNT intrinsic
- GET_COMMAND_ARGUMENT intrinsic
- GET_ENVIRONMENT_VARIABLE intrinsic
- IS_IOSTAT_END intrinsic
- IS_IOSTAT_EOR intrinsic
- parameterized derived types
- keywords in derived type constructors
- type specifiers in array constructors
- allocatable character scalars
- allocatable character assignment
- IEEE_FEATURES module
- IEEE_ARITHMETIC module
- IEEE_EXCEPTIONS module
- Keywords in READ and WRITE statements
- Result KIND specifiers in intrinsics
- Array reallocation - MOVE_ALLOC intrinsic

The following F2003 features are planned to be supported with the programming environment release 5.6, which is scheduled for the 4th quarter of 2006:

- Assumed and deferred type parameters
- Asynchronous I/O and WAIT statement
- Stream I/O
- DECIMAL mode in I/O statements
- Rounding mode in I/O statements
- Derived type extension
- Type-bound procedures
- Finalizers
- polymorphic objects
- SELECT TYPE construct

The remaining features to be completed after the programming environment release 5.6 include:

- enhanced initialization expressions
- user derived type I/O control
- ISO character set support
- text encoding selection in I/O

5. BlackWidow Compiler Optimization

The optimizer for the Cray BlackWidow compiler is derived from the Cray X1 compiler. With default compilation, aggressive automatic optimization is performed at both a scalar and vector level. The primary optimization changes from the X1 implementation derive from the hardware differences between the architectures.

The Cray BlackWidow architecture is based on the NV2 ISA specification, which is an extension of the NV1 ISA specification used for the Cray X1. Significant improvements over the X1 from a compiler standpoint include:

- New inclusive-or bit matrix operation
- Full-speed bit matrix multiplication
- Full-speed vector compress
- 32-bit vector compress and gather/scatter
- A maximum vector length of 128 (versus 64)
- Vector mask registers are 128 bits wide
- 32 vector mask registers (versus 8)
- 32 load buffers
- Branch prediction hints
- Vector atomic memory operations

The compiler uses all of these improvements to create higher performing applications. Compiler directives for finer grain control of optimizations are supported, but are generally not needed.

On the Cray X1, an exclusive-or version of bit matrix multiplication is supported. On the Cray BlackWidow, a new variation that uses inclusive-or has been added. This is available through new vector intrinsic operations, and is also being evaluated for automatic use by the compiler for some vector idioms.

On the Cray X1, vector compress and bit matrix multiplication instructions run at half speed. On the BlackWidow, they run at full speed. This results in obvious speed-ups for bit matrix intrinsics, and speed-ups for complicated conditional code due to the vector compress improvements. Vector reduction performance is also improved, as reductions use the vector compress instruction heavily. This is especially noticeable for smaller trip counts, where the final collapse of a vector's worth of data to a single scalar result relies heavily on the speed of vector compression.

The maximum allowable vector length for the Cray BlackWidow has been increased. The compiler automatically takes advantage of this, effectively halving the number of strip-mined iterations it takes to complete a vector loop. Also, by using constant trip count information and information derived by symbolic range analysis, many loops can be proven to have fewer iterations than the vector length. When this is determined at compilation time, the vectorized loop is guaranteed to execute in one chunk, and all control flow associated with the looping structure is removed.

Similarly, the number of vector mask registers has been increased, which allows the compiler to be much more aggressive in optimizing conditional vector code, allowing the pipelining of comparison operations, and improving other optimizations such as the common subexpression elimination of vector mask expressions. Vector mask registers are very expensive to spill, so the larger set allows for greater optimization opportunities.

The BlackWidow architecture supports branch prediction hints. The compiler automatically makes use of these; estimating branch probability based on internal knowledge of control flow, and on external data provided by the *probability* and *loop_info* compiler directives.

Vector atomic memory operations have been added in the Cray BlackWidow architecture. These provide a limited set of operation-in-memory capability for 64-bit integer operations. For a select set of self-modifying operations on data, the actual work is done in the memory system. The compiler sends an address, a stride or vector of indirect indices, and a vector value to the memory system, and it does the rest. The compiler automatically uses vector atomic memory operations for update operations, and for other operations when they are known to be accessing remote data. The supported operations are 64-bit integer bitwise *and*, bitwise *or*, bitwise exclusive *or*, and integer addition.

As an example, the following loop:

```
for ( i = 0; i < 128; i++ ) {
   a[ix[i]] = a[ix[i]] ^ b[i];
}
```

Translates into the assembly code:

```
a04      a00|128
vl       a04
v02,L    [a03,1],m00   ; Load b[i]
v03,L    [a02,1],m00   ; Load ix[i]
[a01,v03] v02,m00,axor,con
```

The *b[i]* and *ix[i]* operands are loaded into vector registers and sent to the vector atomic memory operation. Note that *a[ix[i]]* is never loaded into a register; the operation is performed in the memory system. By using atomic updates, an exact answer is obtained, even if there are repeated values in array *ix*.

On the Cray X1, stores to small data types inhibit vectorization, due to false sharing issues. Vector atomic memory operations provide a hardware-based mechanism to address this. The Cray BlackWidow compiler uses vector atomic memory operations to vectorize stores to 8- and 16-bit data, and supports full vectorization of small data types (*char* and *short* in C, *integer (kind=1)* and *integer (kind=2)* in Fortran), without any restrictions on data alignment.

6. LibSci

The Scientific libraries for Cray's vector systems are provided in LibSci, while the C and Fortran mathematical intrinsics are provided by Libm.

6.1 Cray X1/X1E LibSci

The libraries for the Cray BlackWidow system are based on the Cray X1/X1E libraries, which provides Fortran interfaces for all routines, supporting 32- and 64-bit default data types.

Libm contains single processor support for:

- Scalar mathematical intrinsics, such as EXP, LOG, and SIN

- Vector mathematical intrinsics
- 32-, 64-, and 128-bit real types
- Random number generation
- Other C and Fortran language features

The latest LibSci release is 5.5. It contains single processor routines, distributed memory and parallel routines, as well as shared memory parallel routines. The LibSci single processor routines support:

- Fast Fourier transform (FFT), convolution, and filtering routines
- Basic Linear Algebra Subprograms (BLAS)
- Linear Algebra Package (LAPACK) routines
- Sparse direct solvers

Multiprocessor in a distributed memory environment is supported on:

- FFT routines
- Scalable LAPACK (ScaLAPACK) routines
- Basic Linear Algebra Communication Subprograms (BLACS)

Finally, LibSci also contains four-way shared memory parallel support across a single node for all Level 3 BLAS routines and for the Level 2 BLAS routines sgemv, dgemv, cgemv, and zgemv. This library is implemented with OpenMP, and including the -lompsci option on the link line accesses it.

There is a small set of LibSci routines that can be inlined with the -O inlinelib option. All Level 1 BLAS routines can be inlined as well as some Level 2 BLAS routines (sgemv, dgemv, cgemv, zgemv, sger, dger, cgerc, cgeru, zgerc, and zgeru).

6.2 BlackWidow LibSci

The LibSci for BlackWidow extends the functionality provided on the Cray X1/X1E systems with additional shared memory parallel routines, sparse computational support, and optimisations, as described next.

The LibSci for BlackWidow will contain additional support for the four-way SMP nodes. In addition to parallel BLAS, there will be parallel FFTs, some parallel LAPACK routines (sgetrf, dgetrf, cgetrf, zgetrf, spotrf, dpotrf, cpotrf, zpotrf, ssytrd, dsytrd, csytrd, zsytrd, ssytrd, dsytrd), and parallel sparse direct solvers. These parallel routines will be implemented with OpenMP and will be integrated into LibSci. It will no long be necessary to specify the –lompsci option at link time. At execution time, the shared memory parallel routines will be used if the application is launched to run on the SMP node using the –d option.

There will be sparse BLAS-like routines included in LibSci to support sparse iterative solvers. It is planned that these sparse routines will support the iterative solvers provided in the PETSc and Trilinos software packages. In addition, these sparse routines can be used in user defined iterative solvers.

Finally, the LibSci will be tuned for the BlackWidow architecture. This architecture contains different cache and memory sizes from the Cray X1/X1E architecture. Routines that are memory bandwidth bound will need to be evaluated and improved, if necessary. In addition, the communication in ScaLAPACK and the distributed memory parallel FFTs will be tuned to exploit the fast one-side communication that the BlackWidow architecture provides.

7. Tools

The Etnus TotalView debugger will be available on the Cray BlackWidow system. TotalView represents the state of the art in visual parallel debugger technology. It is a powerful, sophisticated, and programmable tool that lets users debug, analyze, and tune the performance of complex serial, multiprocessor, and multithreaded programs. TotalView will be available with both its command line interface (CLI) and its graphical interface, supporting threads, MPI, SHMEM, OpenMP, C/C++ and Fortran, as well as mixed-language codes. In addition, it provides some basic support for UPC and Co-Array Fortran.

The Cray performance measurement, analysis, and visualization infrastructure will be available on the BlackWidow system. This framework for performance analysis, which consists of the CrayPat Performance Collector and the Cray Apprentice^2 Performance Analyzer, provides an intuitive and easy to use interface for performance tuning of scientific applications on all Cray platforms. The CrayPat performance collector can gather a wide range of performance data by process and by thread, and can generate a variety of reports. Developers can use these capabilities to locate opportunities for improvements in both performance and system resource usage. The Cray Apprentice^2 Performance Analyzer is a graphical user interface for performance debugging that leverages the CrayPat instrumentation. Rather than trying to focus on performance debugging in a general sense, Cray Apprentice^2 delves further into specific performance debugging domains like MPI, OpenMP, or I/O. Cray Apprentice^2, however, attempts to share intuitive graphical components between the various domains to promote ease of use and it identical interface between the Cray product lines.

8. Conclusions

In this paper we presented the programming environment for the Cray BlackWidow vector system, which consists of state of the art compiler tools, and scientific libraries, supporting a wide range of programming models. The BlackWidow programming environment is being designed to help users achieve the highest possible performance from the hardware. Its design focus is on providing an easy to use and optimised follow on to the Cray X1/X1e software.

About the Authors

Dr. Luiz DeRose is a Sr. Principal Engineer and the Programming Environments Director at Cray Inc. He has more than twenty years of experience in HPC software design and development. He has published more than 40 peer-review articles in scientific publications, primarily on programming environment topics. He can be reached at ldr@cray,.com.

Terry Greyzck is a Principal Engineer and the lead of the compiler optimisation group at Cray Inc. He has more than 20 years of expertise in Compiler development. He can be reached at tdg@cray.com.

Dr. Mary Beth Hribar is the manager of the Scientific Libraries group at Cray Inc. She has more than 10 years of expertise in the field. She can be reached at marybeth@cray.com.

Brian Johnson is the manager of the Compiler Front End group at Cray Inc. He has more than 20 years of experience in compiler development. He can be reached at bhj@cray.com

Mark Pagel is the manager of the MPT group at Cray Inc. He has more than 15 years of experience on software support for high performance computing. He can be reached at pags@cray.com

Dr. Howard Pritchard is a Principal Engineer and the technical lead of the message passing team at Cray Inc. He has more than 20 years of experience in software support for high performance computing. He can be reached at howardp@cray.com.

Greg Titus is a Principal Engineer and the technical lead of the runtime group. He has more than 20 years of experience in software support for high performance computing. He can be reached at gbt@cray.com