THE CRAY PROGRAMMING ENVIRONMENT

Get improved application performance with the Cray Programming Environment (Cray PE), a fully integrated software suite with compilers, tools and libraries designed to maximize programmer productivity, application scalability and performance.

About the Cray Programming Environment

This feature-rich, easy-to-use software package facilitates application development, porting and tuning. Its target architecture consists of multiple dissimilar multi-core processor types closely coupled with accelerators. The programming environment is designed to allow easy porting of existing applications with minimal recoding, minimize changes to existing programming models, and simplify developer transition to the new hardware paradigm.

The Cray Programming Environment suite consists of:

- Cray Compiler Environment (CCE)
- Cray Scientific Libraries (LibSci™ and LibSci_ACC)
- Cray Programming Environment Machine Learning Plugin
- Cray Debugging Support Tools (GDB4HPC and CCDB)
- Cray Performance Measurement, Analysis and Porting Tools (CPMAT)
- Chapel™ programming language

The Cray Programming Environment suite is compatible with MVAPICH and Intel MPI.

Cray Compiler Environment

The Cray Compiler Environment (CCE) is the cornerstone innovation of Cray's adaptive computing paradigm. CCE builds on a well-developed and sophisticated Cray technology base that identifies regions of computation that are either sequential scalar, vector parallel or highly multithreaded. It includes optimizing compilers that automatically exploit the scalar, vector and multithreading hardware capabilities of the Cray system. CCE supports Fortran, C and C++.

Cray compiler features include:

- Compliance with C++11 (ISO/IEC 14882:2011), C++14 (ISO/IEC 14882:2014), ANSI/ISO C11 (ISO/IEC 9899:2011) and ANSI/ISO Fortran 2008
- ANSI/ISO TS 29113 for further interoperability of Fortran with C
- Support for Kernighan & Ritchie C, 64-bit programs and corresponding 64-bit libraries, OpenMP 4.5, hybrid programming using MPI across nodes and OpenMP within the node, IEEE floating-point arithmetic and IEEE file formats, and OpenACC 2.0

Cray Scientific Libraries

The Cray Scientific Libraries, including LibSci and LibSci_ACC, are a collection of highly tuned linear algebra subroutines for Cray systems.

Cray® LibSci™ includes:

- Highly optimized Basic Linear Algebra Subroutines (BLAS) – Cray LibSci includes the auto-tuned BLAS library (CrayBLAS) with custom optimizations for selected routines on Cray systems. The CrayBLAS library is optimized extensively using auto-tuning and runtime adaptation. At runtime, CrayBLAS draws on a performance database and provides a tuned kernel for the precise BLAS calling sequence passed to the library. For Cray systems, CrayBLAS is tuned for both serial and SMP execution. Both serial and threaded versions of the BLAS library are provided.
- Linear Algebra Package (LAPACK) – Cray LibSci includes serial numerical linear algebra routines optimized specifically for Cray hardware. The optimizations in Cray LibSci are mainly at the algorithmic level, affecting both single-core and SMP performance.
- Scalable LAPACK (ScaLAPACK) – Cray LibSci provides an optimized ScaLAPACK library for Cray systems based on ScaLAPACK from Netlib. Cray's ScaLAPACK library is a distributed-memory programming model using MPI. It includes parallel communications improvements that are not part of the standard ScaLAPACK distribution, as well as algorithmic improvements that mirror those made for the LAPACK library.
- Iterative Refinement Toolkit (IRT) – IRT is a custom library that allows users of LAPACK and ScaLAPACK to solve linear systems with increased efficiency by using mixed precision iterative refinement. IRT includes a set of routines to help understand advanced performance available using iterative refinement, and a set of wrappers that allow IRT to be used without changing user code wherever possible. For well-conditioned problems, IRT can provide significant performance improvement with few or no code changes. accelerator resources used by their application.

PROGRAMMING ENVIRONMENT OPTIONS FOR CRAY SYSTEMS

- The Cray Programming Environment suite
- Intel® Parallel Studio*
- GNU compilers and debugger
- PGI compiler and tools
- Qualified third-party tools including Rogue Wave TotalView™ and Arm Forge (DDT+MAP), Python and programming languages
- Arm Allinea Studio

**The Intel Parallel Studio development environment includes both the Intel debugger and the GNU debugger (GDB).*

Cray LibSci_ACC is a library that provides accelerated BLAS and LAPACK routines to enhance user application performance by generating and executing autotuned kernels on GPUs. Cray LibSci_ACC also provides a C API to allow pass-by-value semantics for input parameters, improving productivity for C programmers. Cray LibSci_ACC provides both automatic selection of the appropriate GPU or CPU algorithm based on problem size, and manual selection for programmers who want to explicitly manage the accelerator resources used by their applications.

Cray Programming Environment Machine Learning Plugin

The Cray Programming Environment Machine Learning Plugin allows users to achieve optimized scaling and performance across multiple machine learning frameworks, such as TensorFlow™, utilizing gradient descent. The plugin delivers high performance across Cray node architectures. Key benefits include:

- No parameter servers – the plugin automatically determines which nodes to use, removing the burden of figuring out how many to use and where to put them.
- Easy to launch – the user does not need to specify hosts and port numbers for workers.
- Simple to add parallelism – it's easy to start from a serial training script and include the plugin for parallelism.

With the Cray Programming Environment Machine Learning Plugin, data scientists can easily perform deep learning training on a Cray® XC™ series supercomputer — leveraging either CPUs or GPUs — with nearly ideal efficiency to 512 nodes. The plugin is packaged with the Cray® Urika-XC™ 1.1 UP001 release of Open Source Analytics that includes TensorFlow.

Cray Performance Measurement, Analysis and Porting Tools

Cray provides an integrated infrastructure to help application developers port and optimize applications. CPMAT consists of four main components:

- CrayPAT™, a robust measurement and analysis tool
- CrayPAT-lite, a simplified interface to the measurement and analysis tool
- The Cray Apprentice2 visualization tool
- The Cray® Reveal™ porting tool

Cray Performance Analysis Tool (CrayPAT™)

CrayPAT is the primary performance analysis tool for Cray systems. Programs written in Fortran, C or C++ with MPI, OpenMP or OpenACC, or a combination of these programming languages and models, are supported.

CrayPAT allows users to select the functions to be instrumented at different levels within the program, by user function or by function name. It also provides an API for fine-grained instrumentation. Users do not need to modify program source or their makefile to instrument at a function level. CrayPAT uses binary rewrite techniques at the object level to create an instrumented application, which is generated with a single static or dynamic relink.

A general profile of an executable in CrayPAT provides the total time consumed by a program and its functions. In addition, samples can be taken at set intervals or at the overflow of a hardware performance counter. CrayPAT can take samples of the call stack, dynamic heap memory management statistics, the program counter, and system resource usage.

CrayPAT features include:

- Timing and hardware (CPU and GPU) performance counter measurements for Fortran, C and C++ functions with derived metrics
- The ability to collect and show a program's top time-consumers and bottlenecks (for example, load imbalance) for jobs at scale with low overhead and little program perturbation
- Automatic generation of observations and suggestions based on analysis of collected data
- Integrated data collection and presentation of computation, communication, I/O and memory statistics
- Reports that are presented in text, with content can be exported to other formats such as spreadsheets and programs

CrayPAT-lite

CrayPAT-lite is a simplified, easy-to-use version of the Cray Performance Analysis Tool. CrayPAT-lite provides basic performance analysis information automatically, with a minimum of user interaction, and offers information useful to users wishing to explore a program's behavior further using the full CrayPAT tool set.

CrayPAT-lite supports three basic experiments, providing output to stdout at the end of a program's execution:

- Sample profile – a sampling experiment that can include execution time, aggregate MFLOP count, top time-consuming functions and routines, MPI behavior in user functions, and performance observations or hints
- Event profile – a tracing experiment that generates a profile of top functions traced, OpenMP or OpenACC information if applicable, and performance observations and possible rank order suggestions
- gpu – a trace experiment that generates more detailed OpenACC GPU statistics including host and device time, bytes transferred between the host and device, and data transfer times between the host and device

Cray Apprentice2

Cray Apprentice2 is a visualization tool that presents performance data collected by CrayPAT in report and graphical formats. The Cray Apprentice2 graphical user interface (GUI) makes it easy for developers to view graphs and charts that summarize their programs' performance data and helps to quickly assess the type and severity of performance issues. It takes as input the performance file generated by CrayPAT and provides a familiar notebook-style tabbed user interface. The GUI displays a variety of different data panels, depending on the type of performance experiment that was conducted. It includes call-graph-based profile information and timeline-based trace visualization that focuses on MPI communication, as well as host and device activity for GPU-accelerated programs. It also supports traditional parallel processing and communication mechanisms such as MPI, OpenMP and OpenACC.

Since Cray Apprentice2 uses platform-independent data files, it can run efficiently on Windows®, macOS™ and Linux® operating systems — including laptops and desktops. This allows remote users who do not connect to a Cray system through a high-performance network to benefit from the power of the GUI without experiencing long delays due to X-Window System™ network traffic.

Capabilities of the Cray Apprentice2 tool:

- Provides a high-level overview of a program's key performance characteristics, including top time-consuming functions, load imbalance, memory utilization, and a program profile breaking down computation versus communication
- Uses program runtime summary information as well as event traces; records the amount of time spent in execution of each portion of the codeReports statistics summed across all PEs (or on a per-PE basis) for the whole program, as well as for functions and source code regions (user-selected using the CrayPAT API)
- Shows total execution time, synchronization time, subroutine execution time, communication time, and the number of instructions executed
- Works with both optimized and unoptimized code

Cray® Reveal™
The Cray Reveal code restructuring assistant allows developers to take advantage of more powerful nodes by adding additional levels of parallelism when porting applications. It assists with parallelizing complicated loops, such as those that contain calls to functions.

Reveal extends Cray's performance measurement, analysis and visualization technology by combining performance statistics and program source code visualization with Cray compiler optimization feedback. It provides the ability to easily navigate through source code to highlighted dependencies or bottlenecks during the optimization phase of program development or porting. By using the program library provided by CCE and performance data collected by CPMAT, users can navigate through their source code to understand which high-level loops could benefit from OpenMP parallelism. Reveal provides dependency and variable scoping information for those loops and assists users with creating parallel directives.

Cray Reveal has the following capabilities:

- Source code navigation by loop performance, file, function and loop
- Visual version of CCE's listing information
- Visual version of CCE's optimization messages, filtered by category if desired
- Parallelization assistance including automatic variable scoping for a loop or set of loops, scoping results with feedback on issues found, automatic OpenMP parallel directive generation, and the ability to insert new OpenMP directive into source code

Debugger Tools for Cray Systems

The Cray Programming Environment offers debugger support based on industry-standard debuggers and additional debugging tools developed by Cray. Together, these technologies allow users to address debugging problems at a broader range and scale than conventional techniques.

GDB4HPC
GDB4HPC is a GDB-based parallel debugger used to debug applications compiled with Fortran, C and C++ compilers. GDB4HPC enables users to run a traditional scalable debugging session with a command line interface that provides an experience similar to gdb. It allows programmers to either launch an application or attach to an already-running application. GDB4HPC also supports fast-track debugging and comparative debugger technology that enables programmers to compare data structures between two executing applications. It fully supports applications using the MPI, SHMEM, PGAS and OpenMP programming models. OpenMP support is implemented through the use of the gdbmode command. The special mode exposes the gdb command line syntax and allows the use of thread commands.

Cray Comparative Debugger (CCDB)
The Cray Comparative Debugger (CCDB) is Cray's data-centric debugging tool. CCDB features a GUI that extends the comparative debugging capabilities of lgdb, enabling users to easily compare data structures between two executing applications. If the values of the selected data structures diverge, the user is notified that an error may exist. This capability is useful for locating errors that are introduced when applications are modified through code, compiler or library changes, and for application porting between architectures or programming models.

Cray systems also support third-party debuggers such as Rogue Wave TotalView and Allinea DDT.

Chapel Programming Language
The Chapel programming language is designed for productive parallel computing on large-scale systems. Chapel's design and implementation have been undertaken with portability in mind, permitting Chapel to run on multicore desktops and laptops, commodity clusters and in the cloud, in addition to the high-end supercomputers for which it was designed. Chapel's design and development are being led by Cray, but it receives active collaboration and contributors from academia, computing centers, industry and the open-source community.

Chapel supports a multithreaded execution model via high-level abstractions for data parallelism, task parallelism, concurrency and nested parallelism. Chapel's locale type enables users to specify and reason about the placement of data and tasks on a target architecture in order to tune for locality and affinity. Chapel supports global-view data aggregates with user-defined implementations, permitting operations on distributed data structures to be expressed in a natural manner. In contrast to many previous higher-level parallel languages, Chapel is designed around a multiresolution philosophy, permitting users to initially write very abstract code and then incrementally add more detail until they are as close to the machine as their needs require. Chapel supports code reuse and rapid prototyping via object-oriented design, type inference, and features for generic programming. Existing code can be integrated into Chapel programs (or vice versa) via interoperability features.

Key benefits of the Chapel language:

- Parallel – contains first-class concepts for concurrent and parallel computation
- Productive – designed with programmability and performance in mind
- Portable – runs on laptops, clusters, HPC systems and in the cloud
- Scalable – supports locality-oriented features for distributed memory systems
- Open source – hosted on GitHub and permissively licensed