Cray's Programming Environment for Portable Performance and Programmability on Systems with High-Bandwidth Memory

Luiz DeRose

Sr. Principal Engineer

Programming Environments Director

Cray Inc.

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

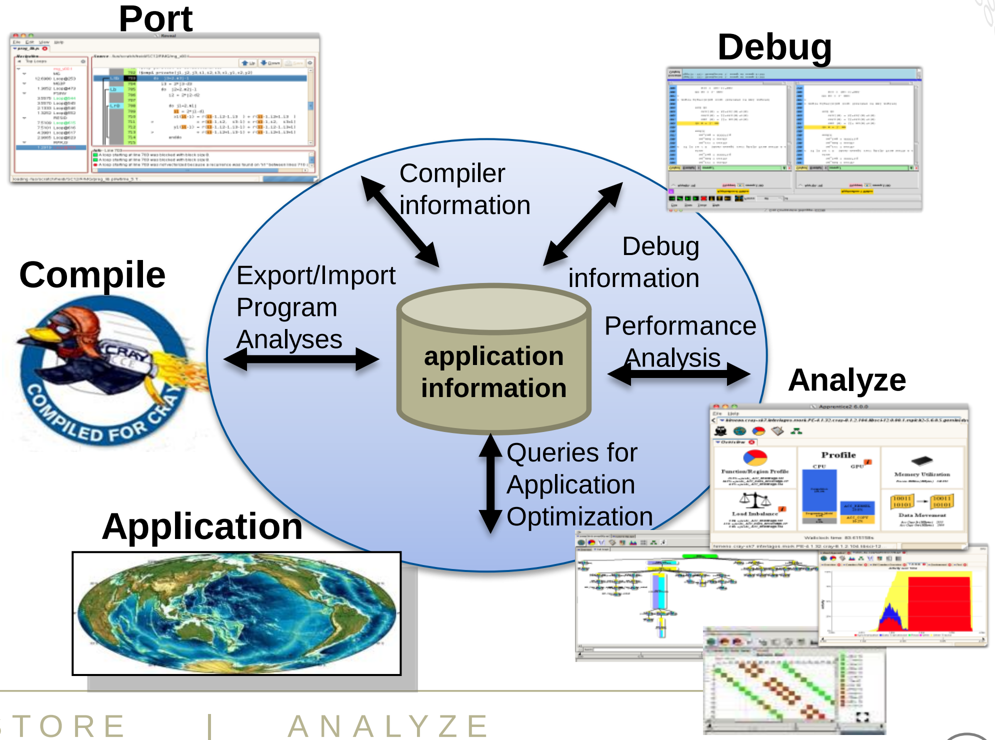
Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Other names and brands may be claimed as the property of others. Other product and service names mentioned herein are the trademarks of their respective owners.

The Cray Programming Environment Mission

- Focus on **Performance** and **Programmability**
  - It is the role of the Programming Environment to **close the gap** between observed performance and achievable performance
- Support the **application development life cycle** by providing a **tightly coupled** environment with compilers, libraries, and tools that will **hide the complexity** of the system
  - Address issues of scale and complexity of HPC systems
  - Target **ease of use** with extended **functionality** and increased **automation**
  - Close **interaction with users**
    - For feedback targeting functionality enhancements



Port

Debug

Compile

Analyze

Application

Compiler information

Debug information

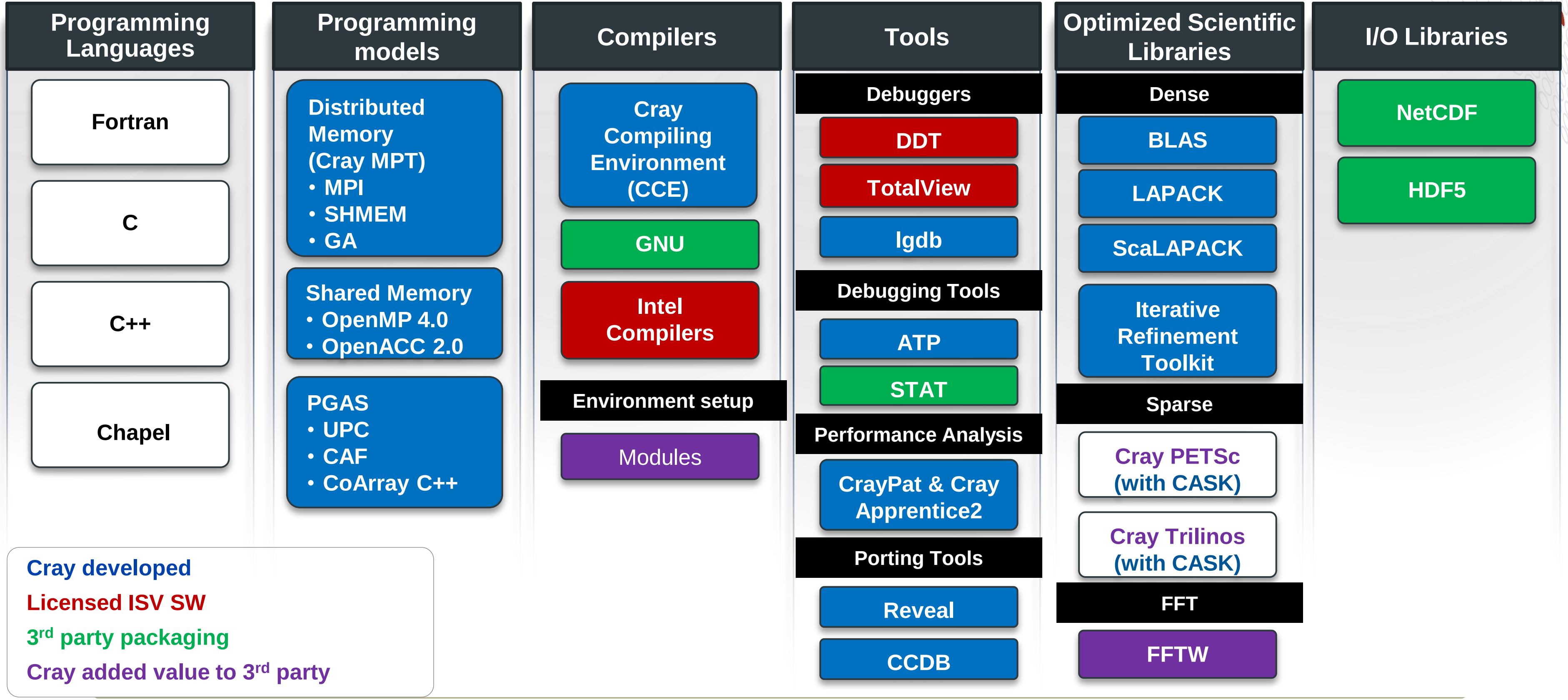
Export/Import Program Analyses

application information

Performance Analysis

Queries for Application Optimization

Cray Programming Environment for KNL



Programming Languages
- Fortran
- C
- C++
- Chapel

Programming models
- Distributed Memory (Cray MPT)
  - MPI
  - SHMEM
  - GA
- Shared Memory
  - OpenMP 4.0
  - OpenACC 2.0
- PGAS
  - UPC
  - CAF
  - CoArray C++

Compilers
- Cray Compiling Environment (CCE)
- GNU
- Intel Compilers

Environment setup
- Modules

Tools

Debuggers
- DDT
- TotalView
- lgdb

Debugging Tools
- ATP
- STAT

Performance Analysis
- CrayPat & Cray Apprentice2

Porting Tools
- Reveal
- CCDB

Optimized Scientific Libraries

Dense
- BLAS
- LAPACK
- ScaLAPACK
- Iterative Refinement Toolkit

Sparse
- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

FFT
- FFTW

I/O Libraries
- NetCDF
- HDF5

Cray developed

Licensed ISV SW

3rd party packaging

Cray added value to 3rd party

CCE Support for High Bandwidth Memory

- **Cray Directive (pragma) to support data allocation in HBM**
  - Provide a directive-only solution
  - Cover more use cases
  - Support for Fortran, C, and C++
    - The directive can be used on **both local and global variables**
      - to place the variables in high bandwidth memory
    - The directive **can also be used on a statement**
      - to change any allocation routines on that statement (allocate, malloc, etc.) to use HBM
    - **If Clause** for dynamic control of directive
    - **Fallback Clause** to control behavior if allocation fails
  - Future direction for memory hierarchy control
    - Ideally will become part of a standard, possibly OpenMP

CCE Proposed API for KNL HBM

- **Directive (pragma) to control placement for high bandwidth memory**
  - Support for Fortran, C and C++
  - Proposed directive
    - **!dir$ memory(*attributes*) *[list of variables]***
    - **#pragma memory(*attributes*) *[list of variables or allocatable members]***
      - *Attributes* – list of desired memory attributes (bandwidth, capacity, nonvolatile, etc.)
        - Initially “bandwidth” is the only allowed attribute
          - Other attributes may be added in the future

- **Statements**
  - Appears prior to an allocation/deallocation statement
  - Changes explicit allocation routines in the next statement to use HBM
    - **Fortran: allocate**
    - **C/C++: malloc, calloc, realloc, posix_memalign, free**
    - **C++: new, delete, new[], delete[]**
      - Directive on deallocation must match (C/C++ only)

CCE Directive for Variable Declarations

```
!dir$ memory(attributes) list-of-vars
#pragma memory(attributes) list-of-vars
```

- **Specified at declaration of variable**
  - For global variables, directive must be visible for every use of global
  - Within type for allocatable members
- **Allowed on:**
  - Local and global variables
  - Scalars, structs and arrays (fixed size and variable length)
  - Fortran allocatables (including members of derived types)
    - Memory allocated will use high bandwidth memory
- **Not allowed on:**
  - Dummy arguments
  - Common blocks or variables within a common block
  - Fortran pointers
  - Variables involved in equivalences
  - Coarray or UPC shared variables

If Clause

```
!dir$ memory(attributes) if(expression)
#pragma memory(attributes) if(expression)
```

- **Dynamic control of directive**

- **For declarations**
  - Expression is evaluated when variable goes into scope

- **For heap allocations**
  - Expression is evaluated when directive is encountered
  - The expression must match on the deallocation (C/C++ only)

Fallback Clause

```
!dir$ memory(attributes) fallback
#pragma memory(attributes) fallback
```

- Controls behavior if allocation fails
- Default behavior: allocation fails
- Fallback behavior: allocation returns normal memory

Cray Memory Directive – Current Status

- **Initial implementation and basic testing of the Cray memory directive is complete for CCE 8.5**
  - Target June 2016 release
    - Support for Intel's FASTMEM attribute is deferred to a future CCE release

- **Internal users are starting to use the feature and providing feedback**

- **Cray is working with OpenMP to incorporate this feature into the OpenMP 5.0 specification (2017/2018)**
  - Cray will present the directive to the OpenMP accelerator subcommittee
  - Intent is to initially include the feature in the annual OpenMP TR by SC'2016