A brief survey of languages for

High Performance Computing.

S. Androutsellis-Theotokis, G. Gousios, K. Kravaritis

**GRNet**

Table of Contents:

[1. Co-array Fortran (CAF) 2](#_Toc274780153)

[2. Unified Parallel C (UPC) 4](#_Toc274780154)

[3. Chapel 6](#_Toc274780155)

[4. X10 8](#_Toc274780156)

[5. HMPP 10](#_Toc274780157)

[6. StarSs 12](#_Toc274780158)

[7. ClearSpeed / Petapath / Cn 14](#_Toc274780159)

[8. PGI compiler with GPU support 16](#_Toc274780160)

[9. Intel Ct and RapidMind (both superseded by the Array Building Blocks) 18](#_Toc274780161)

[10. OpenCL 19](#_Toc274780162)

[11. CUDA 20](#_Toc274780163)

# 1. Co-array Fortran (CAF)

[http://www.co-array.org/](http://www.co-array.org/%22%20%5Ct%20%22_blank)

Introduction

Co-Array Fortran is a small set of extensions to Fortran 95 for Single Program Multiple Data, SPMD, parallel processing.Co-Array Fortran is a simple syntactic extension to Fortran 95 that converts it into a robust, efficient parallel language. It looks and feels like Fortran and requires Fortran programmers to learn only a few new rules.

A coarray Fortran program is interpreted as if it were replicated a number of times and all copies were executed asynchronously. Each copy has its own set of data objects and is termed an image. The array syntax of Fortran is extended with additional trailing subscripts in square brackets to provide a concise representation of references to data that is spread across images.

The Fortran 2008 standard now includes coarrays; the syntax in the Fortran 2008 standard is slightly different from the original CAF proposal.

Features

Coarray Fortran (CAF) is a SPMD parallel programming model based on a small set of language extensions to Fortran 90. CAF supports access to non-local data using a natural extension to Fortran 90 syntax, lightweight and flexible synchronization primitives, pointers, and dynamic allocation of shared data.

An executing CAF program consists of a static collection of asynchronous process images. Like MPI programs, CAF programs explicitly manage locality, data and computation distribution; however, CAF is a shared-memory programming model based on one-sided communication. Rather than explicitly coding message exchanges to obtain off-processor data, CAF programs can directly reference off-processor values using an extension of Fortran 90 syntax for subscripted references. Since both remote data access and synchronization are expressed in the language, communication and synchronization are amenable to compiler-based optimizing transformations.

Uses/Adoption

There were no found no significant applications that use CAF. It is used mainly for scientific purposes. It can and is used in supercomputing.

Its main implementation is provided by Cray Fortran 90 compiler since release 3.1. Another implementation has been developed by the Los Alamos Computer Science Institute (LACSI), at Rice University. They working on an open-source, portable, retargetable, high-quality Co-Array Fortran compiler suitable for use with production codes.

Additional Information

[http://caf.rice.edu/index.html](http://caf.rice.edu/index.html%22%20%5Ct%20%22_blank)

[http://en.wikipedia.org/wiki/Co-array\_Fortran](http://en.wikipedia.org/wiki/Co-array_Fortran%22%20%5Ct%20%22_blank)

Robert W. Numrich and John Reid. Co-Array Fortran for parallel programming. ACM SIGPLAN Fortran Forum Archive, 17:1–31, August 1998.

C. Coarfa, Y. Dotsenko, J. Eckhardt, and J. Mellor-Crummey. Co-array Fortran performance and potential: An NPB experimental study. In 16th International Workshop on Languages and Compilers for Parallel Processing (LCPC), October 2003.

John Mellor-Crummey, Laksono Adhianto, William Scherer III, and Guohua Jin, A New Vision for Coarray Fortran, Proceedings PGAS09, 2009

# 2. Unified Parallel C (UPC)

<http://upc.gwu.edu/>

Introduction

Unified Parallel C (**UPC**) is an extension of the C programming language designed for high performance computing on large-scale parallel machines. The language provides a uniform programming model for both shared and distributed memory hardware. The programmer is presented with a single shared, partitioned address space, where variables may be directly read and written by any processor, but each variable is physically associated with a single processor. UPC uses a Single Program Multiple Data (SPMD) model of computation in which the amount of parallelism is fixed at program startup time, typically with a single thread of execution per processor.

In order to express parallelism, UPC extends ISO C 99 with the following constructs:

* An explicitly parallel execution model
* A shared address space
* Synchronization primitives and a memory consistency model
* Memory management primitives

Features

Under UPC, memory is composed of a shared memory space and a private memory space. A number of threads work independently and each of them can reference any address in the shared space, but only its own private space. The total number of threads is **THREADS** and each thread can identify itself using **MYTHREAD**, where **THREADS** and **MYTHREAD** can be seen as special constants. The shared space, however, is logically divided into partitions each with a special association (affinity) to a given thread. The idea is to make UPC enable the programmers, with proper declarations, to keep the shared data that will be dominantly processed by a given thread associated with that thread. Thus, a thread and the data that has affinity to it can likely be mapped by the system into the same physical node.

Since UPC is an explicit parallel extension of ISO C, all language features of C are already embodied in UPC. In addition, UPC declarations give the programmer control of the distribution of data across the threads. In addition, UPC supports dynamic shared memory allocations. There is generally no implicit synchronization in UPC. Therefore, the language offers a rich range of synchronization and memory consistency control constructs.

Usage/Adoption

There have been found demos and some applications, mainly scientific, which use UPC. Some demos are here: <http://upc.lbl.gov/demos/> and some applications are here: <http://www.upc.mtu.edu/applications.html> .

There are compilers for UPC implemented by Cray, IBM and HP. There are also compilers implemented by UC Berkeley and Michigan Tech. There is also a GCC UPC compiler that extends the capabilities of the GNU GCC compiler.

**License**: Open-source (the exact license type varies for each implementation)

Additional Information

[http://en.wikipedia.org/wiki/Unified\_Parallel\_C](http://en.wikipedia.org/wiki/Unified_Parallel_C%22%20%5Ct%20%22_blank)

[http://upc.lbl.gov/](http://upc.lbl.gov/%22%20%5Ct%20%22_blank)

[http://www.upc.mtu.edu/](http://www.upc.mtu.edu/%22%20%5Ct%20%22_blank)

[http://gccupc.org/](http://gccupc.org/%22%20%5Ct%20%22_blank)

<http://www.alphaworks.ibm.com/tech/upccompiler>

<http://h21007.www2.hp.com/portal/site/dspp/menuitem.863c3e4cbcdc3f3515b49c108973a801/?ciid=c108e1c4dde02110e1c4dde02110275d6e10RCRD>

W. Carlson, J. Draper, D. Culler, K. Yelick, E. Brooks, and K. Warren. Introduction to UPC and Language Specification. CCS-TR-99-157, IDA Center for Computing Sciences, 1999

# 3. Chapel

[http://chapel.cray.com/](http://chapel.cray.com/%22%20%5Ct%20%22_blank)

Introduction

Chapel is a new parallel programming language being developed by Cray Inc. as part of the DARPA-led High Productivity Computing Systems program (HPCS). Chapel is designed to improve the productivity of high-end computer users while also serving as a portable parallel programming model that can be used on commodity clusters or desktop multicore systems. Chapel strives to vastly improve the programmability of large-scale parallel computers while matching or beating the performance and portability of current programming models like MPI.

Features

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. 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.

Usage/Adoption

Chapel is a new language and is not mature enough to be widely adopted. Chapel compiler still considered prototype, i.e. limited use for production environment.

**License**: BSD open-source license

Additional Information

[http://en.wikipedia.org/wiki/Chapel\_(programming\_language)](http://en.wikipedia.org/wiki/Chapel_%28programming_language%29%22%20%5Ct%20%22_blank)

[http://www.prace-project.eu/documents/14\_chapel\_jg.pdf](http://www.prace-project.eu/documents/14_chapel_jg.pdf%22%20%5Ct%20%22_blank)

D. Callahan, B. L. Chamberlain, and H. P. Zima. The Cascade high productivity language. In Proceedings of the Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, pages 52–60. IEEE Computer Society, 2004

B. Chamberlain, D. Callahan, and H. Zima. Parallel programmability and the Chapel language. In Int’l J. High Performance Comp. Apps., volume 21, pages 291–312, Thousand Oaks, CA, USA, 2007. Sage Publications, Inc

SJ Deitz, BL Chamberlain, and MB Hribar. Chapel: Cascade High-Productivity Language An Overview of the Chapel Parallel Programming Model. cug.org

# 4. X10

[http://x10-lang.org/](http://x10-lang.org/%22%20%5Ct%20%22_blank)

Introduction

X10 is a new programming language being developed at IBM Research in collaboration with academic partners. The X10 effort is part of the IBM PERCS project (Productive Easy-to-use Reliable Computer Systems) in the DARPA program on High Productivity Computer Systems.

X10 is a type-safe, parallel object-oriented language. It targets parallel systems with multi-core SMP nodes interconnected in scalable cluster configurations. A member of the Partitioned Global Address Space (PGAS) family of languages, X10 allows the programmer to explicitly manage locality via Places, lightweight activities embodied in async, constructs for termination detection (finish) and phased computation (clocks), and the manipulation of global arrays and data structures.

Features

X10 is designed specifically for parallel programming using the partitioned global address space (PGAS) model. A computation is divided among a set of places, each of which holds some data and hosts one or more activities that operate on those data. It supports a constrained type system for object-oriented programming, as well as user-defined primitive struct types; globally distributed arrays, and structured and unstructured parallelism. [2]

X10 uses the concept of parent and child relationships for activities to prevent the lock stalemate that can occur when two or more processes wait for each other to finish before they can complete. An activity may spawn one or more child activities, which may themselves have children. Children cannot wait for a parent to finish, but a parent can wait for a child using the finish command.

Usage/Adoption

The language is new and is still evolving. Its previous implementation was described as experimental.

**License**: Eclipse Public License

Additional Information

[http://en.wikipedia.org/wiki/X10\_(programming\_language)](http://en.wikipedia.org/wiki/X10_%28programming_language%29%22%20%5Ct%20%22_blank)

[http://www.cs.purdue.edu/homes/xinb/cx10/CX10Report/](http://www.cs.purdue.edu/homes/xinb/cx10/CX10Report/%22%20%5Ct%20%22_blank)

[http://www.prace-project.eu/documents/15\_x10\_wl.pdf](http://www.prace-project.eu/documents/15_x10_wl.pdf%22%20%5Ct%20%22_blank)

Charles, P., Donawa, C., Ebcioglu, K., Grothoff, C., KIELSTRA, A., Sarkar, V., and Praun, C. V. X10: An object-oriented approach to non-uniform cluster computing. In Object-Oriented Programming, Systems, Languages & Applications (OOPSLA) (Oct. 2005), pp. 519–538

Vijay Saraswat et al. The X10 language specification. Technical report, IBM T.J. Watson Research Center, 2010

# 5. HMPP

**Site:**

[http://www.caps-entreprise.com](http://www.caps-entreprise.com/)
<http://www.caps-entreprise.com/fr/page/index.php?id=49&p_p=36>

HMPP allows rapid development of GPU accelerated applications. It is a workbench offering a high level abstraction for hybrid programming based on C and FORTRAN directives. It includes:

* A C and Fortran compiler,
* Data-parallel backends for NVIDIA CUDA and OpenCL, and
* A runtime that makes use of the CUDA / OpenCL development tools and drivers and ensures application deployment on multi-GPU systems.

*[Figure from http://www.caps-entreprise.com/upload/ckfinder/userfiles/images/hmpp\_archi(1).jpg]*

Software assets are kept independent from both hardware platforms and commercial software. By providing different target versions of computations that are offloaded to the available hardware compute units, an HMPP application dynamically adapts its execution to multi-GPUs systems and platform configuration, guaranteeing scalability and interoperability.

HMPP Workbench is based on OpenMP-like directive extensions for C and Fortran, used to build hardware accelerated variants of functions to be offloaded in hardware accelerators such as Nvidia Tesla (or any Cuda compatible hardware) and AMD FireStream. HMPP allows users to pipeline computations in multi-GPU systems and makes better use of asynchronous hardware features to build even better performing GPU accelerated applications.

With the HMPP target generators one can instantaneously prototype and evaluate the performance of the hardware accelerated critical functions. HMPP code is considered to be efficient, portable, and easy to develop and maintain.

HMPP uses codelet/callsite paired directives: codelet for routine implementation and callsite for routine invocation. Unique labels are used for referencing them.

Supported platforms: GPUs, all NVIDIA Tesla and AMD ATI FireStream

Supported compilers: Intel, GNU gcc , GNU gfortran, Open64, PGI, SUN

Supported Operating systems: Any x86\_64 kernel 2.6 Linux with libc, g++, Windows

**Usage / adoption:**

The HMPP directives have been designed and used for more than 2 years by major HPC leaders.

CAPS and PathScale (a provider of high performance AMD64 and Intel64 compilers ) have jointly started working on advancing the HMPP directives as a new open standard. They aim to deliver a new evolution in the General-Purpose computation on Graphics Processing Units (GPGPU) programming model.

**Licensing:**Not free, commercial and educational licenses.

**Additional info:**

PRACE seminar: <http://www.prace-project.eu/news/prace-hosted-a-seminar-on-cuda-and-hmpp>

<http://www.caps-entreprise.com/upload/ckfinder/userfiles/files/caps_hmpp_ds.pdf>
<http://www.caps-entreprise.com/fr/page/index.php?id=49&p_p=36>
<http://www.hpcprojects.com/products/product_details.php?product_id=621>
<http://www.drdobbs.com/high-performance-computing/225701323;jsessionid=HEWGJCK1MESBBQE1GHOSKHWATMY32JVN>
<http://www.ichec.ie/research/hmpp_intro.pdf>
<http://www.prace-project.eu/news/prace-hosted-a-seminar-on-cuda-and-hmpp>

# 6. StarSs

**Site:**

Barcelona Supercomputer Center (Centro Nacional de Supercomputacion)

[http://www.bsc.es/](http://www.bsc.es/plantillaG.php?cat_id=385)

The StarSs programming model exploits task-level parallelism based on C/Fortran directives. It consists of a few OpenMP-like pragmas, a source-to-source translator, and runtime system that schedules tasks to execution preserving dependencies among tasks. Instantiations of the StarSs programming model include:

GRIDSs and COMPSs:

Tailored for Grids or clusters. Data dependence analysis based on files. C/C++, Java

COMP Superscalar (COMPSs) is a new version of GRID Superscalar that aims to ease the development of Grid applications. It exploits the inherent parallelism of applications when running in the Grid. The main objective of COMP Superscalar is to keep the Grid/Cluster as transparent as possible to the programmer. With COMP Superscalar, a sequential Java application that invokes methods of a certain granularity (tasks) is automatically converted into a parallel application whose tasks are executed in different resources of a computational Grid/Cluster. COMPSs also offers a binding to C.

CellSs:

Cell Superscalar (CellSs) addresses the automatic exploitation of the functional parallelism of a sequential program through the different processing elements of the Cell BE architecture. Based on a simple annotation of the source code, a source to source compiler generates the necessary code and a runtime library exploits the existing parallelism by building at runtime a task dependency graph. The runtime takes care of the task scheduling and data handling between the different processors of this heterogeneous architecture. A locality-aware task scheduling has been implemented to reduce the overhead of data transfers.

SMPSs:

While Grid Superscalar and Cell Superscalar address parallel software development for Grid environments and the Cell processor respectively, SMP Superscalar is aimed at ”standard” (x86 and like) multicore processors and symmetric multiprocessor systems.

SMP superscalar (SMPSs) addresses the automatic exploitation of the functional parallelism of a sequential program in multicore and SMP environments. The SMPSs programming environment consists of a source to source compiler and a supporting runtime library. The compiler translates C code with the aforementioned annotations into common C code with calls to the supporting runtime library. Then it compiles the resulting code using the platform C compiler. Tailored for SMPs or homogeneous multicores, Altix, JS21 nodes, Power5, Intel-Core2. C or Fortran.

GPUSs
The programming model introduced by StarSs and extended by GPUSs allows the automatic parallelization of sequential applications. A runtime system is in charge of using the different hardware resources of the platform (the multi-core general-purpose processor and the GPUs) in parallel to execute the annotated sequential code. It is responsibility of the programmer to annotate the sequential code to indicate that a given piece of code will be executed on a GPU.

GPUSs basically provides two OpenMP-like constructs to annotate code. The first one, directly inherited from StarSs, is used to identify a unit of work, or task, and can be applied to tasks that are just composed of a function call, as well as to headers or definitions of functions that are always executed as tasks. The second construct follows a recent proposal to extend the OpenMP tasking model for heterogeneous architectures, and has been incorporated in GPUSs.

**License:**Distributed in source code form and must be compiled and installed before using it. The runtime library source code is distributed under the LGPL license and the rest of the code is distributed under the GPL license.

**Additional info:**

<http://www.hipeac.net/system/files/gpuss.pdf>
<http://www.ogf.org/OGF28/materials/1987/03%2B-%2Bservicess_ogf28.ppt><http://www.bsc.es/media/3825.pdf>
<http://www.bsc.es/plantillaG.php?cat_id=547>

# 7. ClearSpeed / Petapath / Cn

**Site:**<http://www.clearspeed.com/>

ClearSpeed produces computational accelerators for HPC computing, including the CSX600 and CSX700 chips, and the “Advance” full-size PCI-X card that sports two CSX600 chips. The CSX architecture is a family of processors based on ClearSpeed’s multi-threaded array processor (MTAP) core. CSX processors can be used as application accelerators, alongside general-purpose processors such as those from Intel or AMD.

Unlike the GPUs, the ClearSpeed processors were made to operate on 64-bit floating-point data from the start and full error correction is present in the ClearSpeed processors. Furthermore, a Control & Debug unit present in an MTAP enables debugging within the accelerator on the PE level. This is a facility that is missing in GPUs and the FPGA accelerators.

Petapath, a spin-off of ClearSpeed especially for HPC, markets the so-called Feynman e740 and e780 devices. These units pack 4, resp. 8 e710 cards in one unit and can be connected by high-speed PCI Express ( 16× Gen. 2 PCIe at 8 GB/s) to a host processor. There is another feature that is peculiar to the e720 card: its power consumption is extremely low

The ClearSpeed Cn language is based on ANSI C with extensions to support the data-parallel architecture of the CSX processors. The main addition to standard C is the definition of mono (scalar) and poly (parallel) data types. The qualifier poly implies that each PE has its own copy of a value. For example, the definition poly int X; implies that, on the CSX600 with 96 PEs, there exist 96 copies of integer variable X, each having the same address within its PE’s local storage.

The ClearSpeed Software Development Kit (SDK) allows developers to write code to utilize the acceleration of the Advance boards. It consists of:

* ANSI C-based optimizing compiler for the CSX600 and CSX700
* Macro assembler
* Linker, dynamic loader, debugger, Profiler, Eclipse IDE and other tools.
* Various standard C libraries (most include support for both mono and poly data)

PRACE has evaluated prototypes based on Petapath/Clearspeed, including a system composed of ClearSpeed/PetaPath accelerator boards together with the ClearSpeed programming language Cn.

At the Netherlands Computing Facility in Amsterdam, Petapath and HP delivered a power-efficient system, built on eight HP SL170 servers and next generation accelerator prototypes. The system achieves a peak performance of 10 teraflop/s double precision, which is equivalent to more then 60 conventional servers. The system consumes only 6kW of power.

At the CINES supercomputing centre in Montpelier, France, Petapath incorporated the ClearSpeed accelerator technology into a conventional cluster designed by SGI and increased its performance by 50%, with only a 10% increase in power dissipation.

Each technology and architecture is currently being assessed with regard to peak performance/efficiency; programmability; energy efficiency; density; cooling and cost.

WP8.1.10 report on Clearspeed on Petapath concludes:

Future for Petapath looks rather dim:

* ClearSpeed does not make haste with a successor of the CSX700 chip.
* The two-stage data transfer makes it more difficult to achieve optimal performance then users expect.
* The Cn programming model is fairly different from common experience.

**License:**

The SDK is provided with a single-user floating license. Time limited evaluation licenses are also available. No license is required, and no royalties payable, on any software developed with the SDK. The Cn standard libraries are licensed under the terms of the GNU LGPL or similar terms and any software linked with those libraries must comply with those license terms.

**Additional Info:**

<http://insidehpc.com/2010/03/01/prace-looks-at-clearspeed/>
<http://www.clearspeed.com/products/sdk_details.php>
<http://view.eecs.berkeley.edu/wiki/ClearSpeed_CSX600>
<http://developer.clearspeed.com/resources/archives/csug07/ClearSpeed_Software_Tool_Chain-FLYES.pdf>
<http://www.petapath.com/content/prace.html>
<http://www.phys.uu.nl/~steen/web09/clearspeed.php>

# 8. PGI compiler with GPU support

**Site:**

[http://www.pgroup.com](http://www.pgroup.com/)

The Portland Group, Inc (PGI) is a long-time provider of compilers that focus on the HPC user community. PGI 2010 includes the PGI Accelerator Fortran and C99 compilers supporting x64+NVIDIA systems running under Linux, Mac OS X and Windows; PGFORTRAN and PGCC accelerator compilers are supported on all Intel and AMD x64 processor-based systems with CUDA-enabled NVIDIA GPUs.

CUDA is the architecture of the NVIDIA line of GPUs. Currently, the CUDA programming environment is comprised of an extended C compiler and tool chain, known as CUDA C. CUDA C allows direct programming of the GPU from a high level language. Third party wrappers are also available for Python, Perl, Fortran, Java, Ruby, Lua, and MATLAB. The PGI compiler includes support for CUDA Fortran on Linux, Mac OS X and Windows.

GPU designs are optimized for the computations found in graphics rendering, but are general enough to be useful in many data-parallel, compute-intensive programs common in high-performance computing (HPC). CUDA supports four key abstractions: cooperating threads organized into thread groups, shared memory and barrier synchronization within thread groups, and coordinated independent thread groups organized into a grid. A CUDA programmer is required to partition the program into coarse grain blocks that can be executed in parallel. Each block is partitioned into fine grain threads, which can cooperate using shared memory and barrier synchronization. A properly designed CUDA program will run on any CUDA-enabled GPU, regardless of the number of available processor cores

The PGI Accelerator Programming model does “for GPU programming what OpenMP did for thread programming.” Programmers need only add directives to C and Fortran codes, and the compiler does the rest (but one may still need to dig in there and help things along to get the best performance).

The advantages of the PGI Accelerator Model include:

* Minimal changes to the language – directives/pragmas, in the same vein as vector or OpenMP parallel directives
* Minimal library calls – usually none
* Standard x64 toolchain – no changes to makefiles, linkers, build process, standard libraries, other tools
* Binaries will execute on any compatible x64+GPU hardware system
* PGI Unified Binary Technology – ensures continued portability to non GPU-enabled targets
* One Cross-platform HPC Development Environment
* One Integrated Suite of Parallel Compilers & Tools

However using tools like CUDA on NVIDIA’s GPUs requires substantial effort on the part of application developers who now must explicitly manage the transfer of data to the processors of the GPU, fetching of the answer from the GPU, and restructuring of operations to take advantage of the various levels of parallel processing within the hardware (both vector and multiprocessor). OpenCL has the potential to be supported cross-platform, while CUDA is limited to NVIDIA products.

**Applications:**

PGI is the compiler-of-choice among many popular performance-critical applications used in the fields of geophysical modeling, mechanical engineering, computational chemistry, weather forecasting, and high-energy physics. Leading commercial applications built with PGI compilers and tools include ANSYS, ADINA, AVL Fire, POLYFLOW, STAR-CD, LS-DYNA, RADIOSS, PAM-CRASH and GAUSSIAN. Leading community research applications including AMBER, BLAST, CAM, CHARMM, GAMESS, MCNP5, MM5, MOLPRO, MOM4, POP and WRF2 are built and tested by PGI with each release of the PGI compilers and tools.

With companies integrating GPU hardware into their solutions, and other companies developing tools to make the GPUs themselves easier to use, GPUs are starting to benefit from a real network effect.

**License:**

Proprietary

**More information:**

<http://www.pgroup.com/lit/presentations/pgi-acc-ieee.pdf>
<http://insidehpc.com/2009/07/20/pgi-compiler-9-x64-gpu-hybrid-programming/>

# 9. Intel Ct and RapidMind (both superseded by the Array Building Blocks)

**Site:**

http://software.intel.com/en-us/articles/intel-array-building-blocks/

Array Building Blocks (Abb) is a vector programming library, that was developed by merging technologies developed by RapidMind (dynamic compilation for parallel architectures) and Intel Ct (containers and operations).

Abb is a combination of a C++ vector library (full with standard containers and algorithms) with a dynamic optimising runtime library. The containers resemble standard C++ STL containers, although they do not share their interfaces. The runtime library is quite unique; it can dynamically compile and optimise any C++ function (with certain restrictions) to the underlying CPU, if it uses Abb datatypes for input. Moreover, it will arrange for loading the appropriate amounts of data to fill the processor's cache, optimise for

various vector processing instruction sets etc.

The RapidMind implementation of Abb's JIT compiler allowed for code to be run on GPUs or even more exotic architectures such as the Cell BE. Abb is currently limited to Intel CPUs.

**Applications:**

Abb is a new offering and consequently no application is using it. It could theoretically be used as a generic replacement for vector datatypes and operations in C++ programs.

Abb has not been released to the market yet. RapidMind's product that preceded it has seen quite significant press exposure and fair product use. Notable programs that used RapidMind are various games on the PS3 console, the RTT raytracer and others. No use has been identified on HPC environments.

**License:** Proprietary

# 10. OpenCL

**Site:**

<http://www.khronos.org/opencl/>

OpenCL is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors. OpenCL includes a language (based on C99) for writing kernels (functions that execute on OpenCL devices), plus APIs that are used to define and then control the platforms. OpenCL provides parallel computing using task-based and data-based parallelism.

One of the unique characteristics of OpenCL is that it allows for code to target different processing unit types (CPUs or GPUs) dynamically, depending on the host configuration.

OpenCL consists of modest extensions to the C language, which includes support for vector types, managed data buffers, mathematical consistency guarantees across computing devices and a limited number of new keywords.

An OpenCL compute platform consists of several compute devices, where a compute device is either a CPU or a GPU. A compute device has one or more compute units (e.g. a dual core CPU has 2 compute units). An OpenCL application submits work to compute devices, wrapped in work items called kernels (C functions that perform a certain computation). Unlike traditional C code, OpenCL kernels are incorporated into the application in an uncompiled state. They are compiled on the fly and optimized for the user’s hardware before being sent to the GPU for processing. Each compute device maintains a queue of kernels, which can be executed in order or out of order based on the invocation of external signals. A kernel reads data from a private memory area setup before its invocation, performs the computation and copies data back to a special result area.

The OpenCL runtime is responsible for a number of tasks such as setting up memory buffers, scheduling work on the available compute units and (optionally) compiling the kernels to better match the underlying architecture.

**Applications:**

No major application, except from impressive demos, has been found that uses OpenCL. Also, no uses on supercomputers have been discovered, although support for running clusters of OpenCL compute nodes can been enabled through the Mosix Virtual OpenCL project. Other creative uses of OpenCL include GPU assisted password cracking and malware applications. (check out paper GPU assisted malware)

OpenCL has been developed by Apple and was standardised by the Khronos group. Since then it has received wide industry adoption, mainly in the graphics processing community. All major graphics vendors offer OpenCL enabled drivers. Apple MacOSX includes OpenCL for CPUs and GPUs since version 10.6. IBM provides an OpenCL implementation for Cell BE processors running in Bladecenters. Intel will support OpenCL in future CPU architectures + compilers.

**License:** Open Standard, License depends on implementation

# 11. CUDA

**Site:**

<http://www.nvidia.com/cuda>

CUDA is the computing engine in NVIDIA graphics processing units (GPUs) that is accessible to software developers through variants of industry standard programming languages.

CUDA's aim is to expose the massive data parallel processing power of GPUs to generic computation problems. The CUDA model consists of a CPU that handles generic computation and one of more GPUs that handle specially marked portions of the code called kernels. Kernels are written in a C-like language, which is then compiled to assembly suitable for execution on a GPU. Each CUDA execution engine consists of a large number of execution units, which in turn supports a large number of on the fly threads. The typical execution of a kernel involves allocating memory on the graphics device, copying data from main system memory to it, firing up the algorithm execution and then copying the results back. There are several intricacies that CUDA hides, for example splitting the load among thread groups and synchronising threads at predefined points.

**Applications:**

CUDA has been adopted by a wide arrange of applications in various fields. Being one of the first (debuted in 2007) technologies that allowed generic GPU programming, its use has been widespread in several scientific, medical, financial and consumer products. NVidia produces machines (rack-mounted and workstations) that are able to execute massively parallel computations efficiently. The second largest supercomputer in the world (Nebulae, China) sports CUDA processors, an indication that CUDA might be ready for HPC in the real world.

CUDA can act as a base layer for many other technologies, like C++ container libraries (Thurst and Thurst Graph), mathematical libraries (CUBLAS), bioinforatics (gromacs) and others. OpenCL has also been implemented on top of CUDA.

CUDA is probably the most widely adopted GPU acceleration architecture. To that end, the availability of free (and high quality) software tools and the fact that the target audience is quite big (NVidia is already the largest graphics card maker) led to a large number of developers using it. CUDA runtime bindings exist in several languages, so one can use CUDA indirectly, though a scripting language such as Python. However, CUDA is not an industry supported standard and no offerings outside NVidia exist, so implementations are tied to a single provider.

**License:** Proprietary