MSR Accelerator v2 Update

October 24, 2011 – READY FOR EXTERNAL RELEASE

# Introduction

Today’s breadth of programming models for software development provides a large number of options that do not require a developer to have deep knowledge and experience in programming to the different specific Instruction Set Architectures (ISA) and other programmable resources that are ubiquitous in today’s computing platforms. Software developers can write well behaved code that runs across a wide range of devices—with reasonable expectations for performance and stability—due to the level of ISA target abstraction available and supported in their development tools and compilers. The developers chosen compiler manages the diversity of targets, and manages performance tuning per target at compilation time.

From the programmer’s perspective, this is the desired programming model. However programmers are facing more and more demands to meet the performance requirements of today’s business, using resources that are not yet programmable through the elegant programming abstractions they expect and rely upon.

The growth in interest and increasing numbers of application designs in the areas of multi core computing and General Purpose GPU computing creates significant barriers for compiler and developer tool creators to delivering diverse ISA target abstractions. The challenges faced by the creators of software development tools amplifies the challenges and complexity that main stream software programmers must face to deliver applications that take advantage of the resources that are present but often go virtually untapped.

Parallel computing has seen progress in several areas that enable programmers to access and utilize some of these resources. However, most solutions being offered today have their own sets of tradeoffs and usually only address a small segment of the general developer population. The challenge remains to overcome these barriers and deliver tools that continue to deliver rich abstractions suitable for mainstream application developers that can extract the best possible performance from the broadest possible set of resource targets that are considered today as key assets for heterogeneous computing.

# Microsoft Research Accelerator

# Accelerator’s Scope

Heterogeneous Parallel Computing

Accelerator addresses one of three segments of the broad parallel computing problem space – “embarrassingly parallel” data-parallel algorithms. Accelerator is a high-level data-parallel programming library designed to accelerate code execution. It provides developers with a consistent model for accessing the available resources that are already present in today architectures in a way that abstracts and manages vendor specific implementations and unique capabilities.

Accelerator enables users to run compute-intensive code with predictable performance and reliability across a large installed base of computing devices while also ensuring portability to future ISA’s.

Extend the existing programming model

Accelerator provides access to new resources and capabilities without requiring programmers to learn new languages or programming models. For example, programmers can access GPU resources without knowing anything about high-level shader language (HLSL).

Let the programmer decide

Accelerator is designed to address a key issue every programmer faces when abstraction inherently reduces resource performance and capabilities to a lowest common denominator. Accelerator’s design team made a somewhat novel decision to quickly find the right “balance point” between abstraction and performance. This decision became a key design tenant for Accelerator: “Let the programmer decide”.

Extensible Target implementations

Accelerator supports pluggable 3rd-party developed targets and enables programmers to code to specific targets. The targets fall into two general categories: hardware and programming.

*Examples of hardware targets:*

* CPU ISA’s X86 and x64
* Direct X 9 and 11 class GPUs
* FGPA’s
* DSP’s (not yet prototyped)

*Examples of programming targets:*

* Distributed Heterogeneous Computing, such as the Azure programming model
* CUDA
* OpenCL (not yet prototyped)

*ccelerator is designed to support Vendor-specific hardware target implementations*

Language independence

# The Accelerator programming model is implemented as a library that can be used with any Windows programming language that supports linking with a C calling interface. Applications that use Accelerator have been written in C++, C#, F# Fortran90[[1]](#footnote-1) and Haskell. Well documented sample code distributed with Accelerator is written in C++ and C#.

Programming Model

New capabilities for Accelerator are under development as we extend its capability. Among the key features being assessed for upcoming releases are:

* Enhanced support for data including user defined data structures.
* Full GPU data type support.
* Utilization of the full capabilities of supported hardware and ISA’s.

To achieve the best performance on supported targets, we are also interested in support for pre-compilation, which separates the JIT-ing phase from execution.

# Additional Accelerator overview

If you are just learning about Accelerator, an excellent overview that will give you more details on the design and usage of Accelerator is; *“Computing Without Processors”* Satnam Singh; Communications of the ACM Vol. 54 No.8, Pages 45-54 <http://queue.acm.org/detail.cfm?id=2000516>

# Microsoft Research and Accelerator

Accelerator is a Microsoft Research development tool, built “in-house”, to address a specific set of challenges faced that are faced daily by our researchers and the community of their research peers during the development phase of research prototypes used for experiment data collection and the validation phases of research projects. Subsets of research projects also need to simplify the ability of other researchers to duplicate published experimental results that Accelerator is a candidate to address.

There is sufficient critical mass around the development challenges in these areas and the population of researchers outlined here, to lead MSR to undertake the development of Accelerator.

The goals for Accelerator required that we simplify the decision process around design, features, capabilities and prioritization. When trying to determine the needs of the broad and rather diverse set of potential users, the process for determining Accelerator’s the features and capabilities and prioritizing the implementation needs to remain focused. In support of this goal, we decided that Accelerator itself is the vehicle, our release cycle is the method, and user feedback will determine the next set of improvements and milestones our users want.

A note about the MSR “Not for Commercial Use” EULA

We have received a lot of feedback and questions regarding Accelerator’s EULA.

As a general rule, licensing of Microsoft technologies for commercial use is offered only through our commercial product divisions. The purpose of this general rule is to ensure that any company that makes a bet on using a particular technology, feature or product that they license from Microsoft, also has a reasonable and predictable set of expectations for how Microsoft will maintain, support, and improve what is under license.

In some cases, MSR will work directly with commercial 3rd party companies interested in pursuing a full evaluation of technologies such as Accelerator with the goal of entering into a commercial use license agreement with Microsoft.

Contact our Accelerator support alias at: [aclsup@microsoft.com](mailto:aclsup@microsoft.com) for further inquiries.

Accelerator design goal for SW developers

1. Developers use a programming and design model they are already familiar with
   * Accelerator supports both managed and unmanaged libraries.
   * Accelerator uses data types based on the **ParallelArray class.**
   * Supported Accelerator operations include standard arithmetic and comparison operations; matrix operations such as transposition, shifting, and reduction; and linear algebra operations such as outer and inner product.
   * Accelerator includes support for specialized primitive operations suchas Fast Fourier Transforms (FFTs), matrix multiplication, and pseudo random number generation[[2]](#footnote-2)
   * The full set of operations is detailed in the Accelerator SDK, which is posted on the download section of http://research.microsoft.com.
2. Well written code will have the best possible performance, without requiring the developer to have the deep expertise and specific knowledge of the architecture that are required to implement performance tuning optimizations.
   * Accelerator generated code will run with predictable performance and reliability across a large installed dbase of computing devices.
3. Language agnostic
   * Accelerator is a library, and is generally accessed via operator overloads from the developer’s language of choice (we currently have C++ and .NET and Fortran90 bindings). The low-level C-callable interface is intended to be easy to invoke from any language that supports calls to native code exposed by name from a DLL. This means Accelerator does not depend on the host language for its type system, math operators, etc.. Interpreted, jitted and compiled languages are all supported.
4. Supports a large population of developers with a range of skills and specific programming needs.
   * Accelerator is designed for users starting with mainstream development, experienced C++ and .Net developers, and extends to the class of developers that are growing their knowledge expertise and skills specific to architecture requirements around code performance tuning optimizations.
5. Support advanced developers who require vendor-specific targets
   * Accelerator design allows developers to use target implementations from hardware vendors and in-house developers can create their own target implementations
6. Help advanced developers “future-proof” target specific designs.
   * Help improve code reuse and agility by allowing developers to “compartmentalize” target specific code in their designs and coding practices.
   * Enables accurate estimates of the costs associated with converting to newer technologies or different vendor solutions as they mature and come to market.

# Accelerator 2.2 Release

**Key feature:** Addition of a CUDA™ target.

Additional improvements / changes include:

* Improves developer control over asynchronous operations through a better method of “wait for completion.”
* Introduces **ParallelArrayParam** and its child classes, which allow reuse of entire expressions with different data sets.
* Includes previously released incremental bug fixes and patches reintegrated to meet design specification.
* Includes miscellaneous “new” Bug fixes.

1. Our Fortran wrappers are a bit “stale”, so we are making them available on request and will assist in updating them. [↑](#footnote-ref-1)
2. Not implemented for 2.2. Implementation has been prioritized based on user demands [↑](#footnote-ref-2)