Intel(R) Array Building Blocks Code Samples

Overview

To help jum pstart your application development, we provide code samples that illustrate the use of Intel® Arra Building Blocks in various workloads including those often used for financial services, graphics, image processi medical imaging and more. The sample applications provide the most direct way to determine:

- Whether the software is working on your system
- How you can use the different language constructs (for example, operators, functions, facilities and so or
- How to write code and create an application

Installation

The code samples are contained in the installation package and by default are installed to

- Windows* directory **C:\Program Files\Intel\arbb\<version>\samples**
- Linux* directory **/opt/intel/arbb/<version>/samples**

where <version> is the version of the product being installed.

Building and Running the Samples

On Windows*, open one of the Microsoft* Visual Studio* solution (.sln) files located in the samples folder. For examples, if you use Visual Studio 2005, double-click the file C:\Program Files\Intel\arbb\<version>\samples \samples-vs05.sln.

1. Select a configuration (the default setting is Debug - Win32 configuration)
2. From the **Build** menu, select **Build Solution** to build the entire solution, which consists of individual proj for each sample. Use the **Solution Explorer** to view projects grouped in folders by category.
3. In the **Solution Explorer**, right-click the name of the sample you are interested in and choose the **Set as StartUp Project** option from the context menu.
4. Click the **Run** button to run the selected sample.

On Linux*, use one of the following shell scripts, located in the folder /opt/intel/arbb/<version>/tools, to build a run the samples:

- build_run-icc.sh - automatically build and run the sample applications using Intel® C++ Compiler
- build_run-gcc.sh - automatically build and run the sample applications using GCC*

Known Issues

- The problem size for some samples (for example, graphics/raytracing1) is too small to get meaningful performance measurement. Users need to manually set a big enough problem size to be able to see a reasonable speedup.

Detailed Description

Sample	**Description**	**Algorithm**	**Implementation**
Category: finance			
binominal-tree	Numerical lattice for pricing European options.	Stream of option pricing evaluations with high arithmetic intensity (**exp**, **sqrt**) .	(1) A **map** to parallelize over pri multiple options. Uses a series o **_for** loops for each time step an calls **replace()** on elements of lo (temporary) containers as well a containers for output of the opti prices.
black-scholes	Analytical method for pricing European options. Optionally evaluates or approximates polynomials.	Data-parallel random number generation using **scan**. Option stream with arithmetic intensity (**ln**, **exp**, **sqrt**).	(1) Uses the **call** operator to inli an Intel ArBB function whose o loop parallelizes over options. Illustrates the use of **select** to choose between two terms durin polynomial evaluation.
monte-carlo	Stochastic method for computing financial options using the Blackscholes formula given randomly varying prices. Can optionally generate the sequence of random numbers using a multiplicative congruential generator (MCG).	Data-parallel random number generation using **scan**. Option stream with arithmetic intensity (**exp**). 1D and 2D accumulation (reductions).	(1) Uses the **call** operator to inv an Intel ArBB function whose o loop parallelizes over options. Generates a normally distributed random sequence using a transformation of a uniform ranc sequence. Uses a nested **_for** loo over prices to perform 1D vecto arithmetic, and uses **add_reduc** and **replace** to accumulate a res (2) Uses **reshape** and **repeat_co** perform an equivalent 2D implementation. Illustrates the u of **add_reduce** for accumulatior

Sample *Trademarks	Description	Algorithm	Implementation
poisson-solver	Monte-Carlo method to solve Poisson functions (MCP solver). Uses a sequence of random numbers from a linear congruential generator (LCG).	Data-parallel random number generation using **scan**. Kernel with nested loops and high arithmetic intensity (**sin**, **cos**). Minimum distance computation using a series of thresholds in the inner loop. Unbalanced load where the number of iterations depends on random input.	(1) Uses the **call** operator to generate a large vector of scalar random numbers. Followed by a **map** over points illustrating nest **_for** and **_while** loops for a ranc walk. The inner loop is a series c **_if** statements to compute a minimum distance. (2) Equivaler implementation using **map** to perform scalar arithmetic.
randomlib	Code that can be in-lined to generate a normally distributed random sequence using the following algorithms: • Linear Congruential Generator (LCG) • Multiplicative Congruential Generator (MCG) • Combined multiple recursive generator with two components of order 3 (MRG) • Generalized feedback shift register generator (R250) Mersenne twister (MT)	Data-parallel random number generation using **scan**. Scan collectives, bitwise operations	(MCG) Uses the **call** operator tc invoke native code stubs from within an Intel ArBB function. T actual native implementation ca switched at link time. (General) Use of **mul_scan** to generate indices. Illustrates the u of **rotate** and **select** on seeds. Illustrates the use of a bitwise & operation (a mask) to simulate a vector modulus operation.
Category: graphics			

Sample	Description	Algorithm	Implementation
raytracing1	A kernel used to create a realistic visualization of a scene when tracing rays from a camera through an image plane to a light source. For each pixel in a 2D array, the kernel determines the closest ray-triangle intersection and evaluates the pixel shade using a lighting calculation. This implementation is brute-force and does not use an accelerator: every ray is compared to every triangle.	For each triangle in the scene, compute the intersection and distance to triangle. Compute the minimum distance and shade the triangle closest to the camera. A simplified lighting calculation is given by a proportional sum of diffuse, specular and ambient light. The ray tracing algorithm is parameterized over 1-component or 2-component inputs and outputs.	(1) Uses the **call** operator over a pixel array. Within the **call** body **_for** loop over the height of the pixel array is performed. For eac row, a **map** over 1D lines of pix performed. Illustrates the use of **index** to generate an arithmetic sequence and **replace_row** to populate rows of the 2D outputs Uses **at** to extract a one-compon **array** from an N-component **ar** (2) An equivalent **map** over a 2I pixel array is performed. Illustra the use of **index** to generate a 2I sequence. **Note:** The bulk of the implementation differs only in th use of scalars versus 2-tuples for positions and directions. Therefo ray tracing is parameterized over different data types. (3) A variation on (2) using 3-component tuples and large vectors of 3-component tuples instead of separate variables (i.e RGB and XYZ). Illustrates the u of **get** and **set** for components of tuple
raytracing2	A variation on raytracing1 where ray-triangle intersection is limited to triangles in grid cells that intersect with rays. In other words, a uniform spatial partition is used for acceleration.	A variation on raytracing1 where the triangles are indexed by a uniform grid accelerator to limit the number of triangle-ray tests that are required.. For each cell in a 3D grid, an initial test is performed to determine if the ray intersects the cell. Ray-triangle intersection is performed only when rays intersect grid cells.	(1-3) See (1), (2) and (3) for raytracing1. **Note** Uses **_break** to perform an early exit from a **_for** loop or a **_whil** loop.

Sample	Description	Algorithm	Implementation
Category: image processing			
convolve	Convolution of a 2D image with a discrete Gaussian function.	A local gather over a fixed neighborhood around each pixel of a 2D image. 1D convolution along X and Y axis of a pre-computed 2D stencil of coefficients. Clamps output to 255 to prevent saturation of the 8-bit unsigned image data. Optionally runs with convolution stencils of 5x5 or 9x9 pixels.	(1) Uses the **call** operator to implement separable convolutio using large vector math. Calls **sh** to perform vector arithmetic on neighbors. Optionally performs a averaging filter. (2) A variation on (1) with manu unrolling rather than **_for** loops separable convolution. Only wo with a 5x5 pixel convolution ste (3) A **map** operation using neste **_for** loops to perform convolutic Calls **num_rows** on the 2D sten to operate on square kernels of a size. (4) This tuned version casts the unsigned image data to single-precision float. Next, a **_for** loop performed over 64-pixel wide st of the image. For each block, ne for loops are used to unroll the convolution. Uses **section** and**replace** to operate on a 64-p wide strip of the image. This is followed by a similar loop to pro the portion of the image that doe not fit into strips of 64 pixels.
gauss-convolve	Convolution of a 2D image with a discrete Gaussian function.	Similar to convolve. Uses different stencil sizes and does not assume odd stencil sizes.	(1) Two **_for** loops to perform separable convolution using larg vector math. Uses **shift_row** anc **shift_col** for the 1D convolution along X and Y axis. (2) Equivalent implementation u **map** to perform scalar arithmeti Illustrates in-lining of multiple C routines (one for each axis) into single Intel ArBB function.

Sample	Description	Algorithm	Implementation
sobel	An edge detection filter for a 2D image that uses the gradient (rate of change) of image intensities.	A gather over a fixed neighborhood around each pixel of a 2D image. Separately computes the gradient along the X and Y axes. This variation on a Sobel filter outputs the largest of the two gradients (with clamping to avoid saturation of 8-bpp image data).	(1) Uses **call** to invoke an Intel ArBB function that in turn inline separate functions to compute th gradient in X and Y using large vector arithmetic. Calls **shift** to perform vector arithmetic on neighbors. (2) Equivalent implementation u **map** to perform scalar arithmeti
Category: medical			
3D-dilate	A morphological operator for dilation applied to 3D grayscale images.	Loops over a neighborhood defined by a 3D binary mask (structuring element). For each neighbor corresponding to a non-zero mask entry, the image is updated with the largest difference between the neighbor and a height field matrix. Features are dilated when voxels neighboring the structuring element are incorporated (assigned similar intensities). The height matrix provides intensities for non-flat structuring elements.	(1) Three nested **_for** loops are u to iterate through the mask. A ca **create** makes a local buffer to st maximums. Calls **shift** to perforn vector arithmetic on neighbors. (**num_cols**, **num_rows** and **num_pages** to operate on a mas arbitrary size.
3D-Erode	A morphological operator for erosion applied to 3D grayscale images.	Similar to 3D-dilate, except that the smallest difference is output.	Similar to 3D-dilate. Uses **min_reduce**.
3D-gauss-convolve	Convolution of a 3D image with a discrete Gaussian function.	Similar to guass-convolve, except that a **3D** convolution stencil is applied to **3D** image data.	Similar to guass-convolve. Uses **shift_page** in addition to **shift_r** and **shift_col** to handle the Z ax
back_projection	A technique for image reconstruction used with inputs from computed axial	A spatially-coherent gather along projections (rays) through each pixel of a 2D	(1) Uses the **call** operator to parallelize over pixels in the 2D output image. Uses a **_for** loop i

Sample	Description	Algorithm	Implementation
	tomography (CAT) scans.	image. Applies the inverse Radon transform to reconstruct a 2D image given a set of projections through that image. Uses 1D interpolation to update the output image with the contribution from the nearest projections. **Note** A simple scan geometry is assumed (radically symmetric 1D orthographic projections rather than a helical scan). In addition, it is assumed that sharpening of sets of input projections (sinograms) has already been performed.	the **call** body to iterate through projection angles. Uses a table lookup to compute the **sin** and **c** of each projection angle. Calls **fl** and **ceiling** on large vectors pric interpolation. Uses the **+=** opera to integrate contributions. (2) A variation on (1) that uses **reshape** to create a 2D product angles and projections rather tha packed 1D vector. Within the **ca** body, the indexing is modified tc perform a 2D gather using a two-component index.
Category: misc			
mandelbrot	Fractal data set generation.	Iteratively applies a quadratic polynomial map over complex numbers and computes it escape time to compute a fractal set.	(1) Uses a **_for** loop in a **map** operator to iteratively refine the output. Uses the complex numbe (using std::complex over Intel A floating-point types) to perform complex multiplication and addi Calls **abs** to compute the comple norm, and uses **_break** to exit ea when the hard-coded bounds are exceeded. (2) An alternative implementatic using a **_for** loop in a call operat Creates a large vector that is loc the Intel ArBB function, comple and 2D. Performs a fixed numbe iterations, and stops updating the output when the fractal bounds l

Sample	Description	Algorithm	Implementation
			been exceeded (this performs m work than the version (1) using a early exit).
spec-samples	Calling code that details the behavior of various Intel ArBB operations on dense and nested containers. The operations are divided into three categories: • Collectives used for reductions and scans. • Facilities for building and querying the structure of nested data containers. • Operations on dense and nested containers, permutation operations on elements or nested segments of containers.	(1) Full and partial collective operations are performed. The partial collectives reduce the dimensionality of the input set rather than returning a single value. Collective operation is illustrated using dense and nested containers. (2) Illustrates the reshaping of dense containers as nested containers, flattening of nested containers, and **split/unsplit/cat** operations. Also shows how to extract sizes of dense containers and nested segments. (...2) Illustrates the creation and initialization of large vectors and index sets. Also shows how to section large vectors and update sections of large vectors. (3) Permutes data using swizzle, pack, shift, rotate, sort and shuffle operations. Many of these operations have inverses, such as pack/unpack.	(1) The calling code, inputs and outputs are detailed for full/parti reductions using **add_reduce,** as well as exclusive and inclusive s (**add_scan** and **add_iscan**). (2) Uses **reshape_nested_lengtl** generate nested vectors from de vectors based on segment descriptors. Couples this operati with a type cast using **reshape_a** Calls **split**, **unsplit** and **cat** with inputs and/or outputs that are ne containers. (...2) Calls **value**, **lengths**, **flags** **offsets** to extract information ab nested containers. (...2) Calls **create** for large vect and illustrates the construction c index sets. Uses section and repl to operate on pieces of large vectors. (3) Performs swizzle, mask, pack/unpack and scatter operati on large vectors using large vect to specify the output indices. (...3) Calls **shift**, **shift_sticky** an **rotate** with options to permute **dense** and **nested** containers bot left and right. Note that full segments of nested containers ca be permuted. (...3) Calls **sort** to perform direc and indirect sorts on dense containers. (...3) Calls **shuffle**/**unshuffle** to perform strided interleave/de-interleave of dense containers. (...3) Shows how to use repeat a

Sample	Description	Algorithm	Implementation
			repeat_row variants to replicate data in dense containers.
Category: seismic			
3dstencil	Convolution used in reverse time migration (RTM).	Convolution using a 7x7x7 cross-shaped kernel.	(1) Uses the **map** operator to perform scalar arithmetic. Uses relative indices to gather values neighbors.
convolution	1D and 2D convolution for a seismic image.	Separable 2D convolution using a cross-shaped kernel.	(X) Uses the **call** operator to implement 1D convolution on th x-axis between a seismic trace a large array of weights. Calls **shif** access neighbors within a **_for** lo to perform convolution with an arbitrarily sized array of weights Uses **create** to generate a large vector output of any specified si (Y) An equivalent operation on t Y axis performed on half of the input data set. (2D) Uses the **call** operator to perform a 2D convolution with a cross-shaped stencil of fixed size Uses **shift_sticky** to perform vec arithmetic with neighbors using a zero-flux assumption for out-of-bounds accesses (clamped to the nearest boundary value). Uses a stride of 2 on the x-axis when gathering neighbors.
kirchhoff	Generic Kirchhoff migration assuming constant velocity of seismic waves through a sub-surface.	Accumulates the contributions of each seismic trace to a sub-surface reconstruction. Uses a constant velocity model where the time from source to receiver is proportional to the distance between the source and receiver. Uses the equation of a circle to	(1) Uses the **call** operator to implement migration with large vector arithmetic. Uses **create** to allocate a large vector output. Constructs sets of **indices**<> wit the user-specified resolution. Us **_for** loop to parallelize over circ centers. Uses a **select** statement perform a boundary check.

Sample	Description	Algorithm	Implementation
		determine the possible reflection points. Uses correlation between multiple source-receiver pairs to identify the location of the reflecting sub-surface.	(2) A 2D variation on (1) where output and index sets are 2D X-: datasets. Uses **repeat_col** and **repeat_row** to generate the 2D index sets. Uses **create** to initial: 2D large vector containing two-component tuples used to perform a gather. Specifically, tl 2-tuples are used to index the tra data to determine the appropriat contribution for the output reconstruction.

Do you need more help?
Click tags links for related articles Search Knowledge Base Visit User Forums Get other Support options

This article applies to: Intel® Array Building Blocks Knowledge Base