CODE MODERNIZATION

Intel® C/C++ Compilers

Set Compiler path and compiler related environment variables

```
$source <install-dir>/bin/compilervars.sh
```

Invoke C compiler

```
$ icc  <C filename >  -o  <Executable filename> <compiler options>
```

Invoke C++ compiler

```
$ icpc  <C++ filename >  -o  <Executable filename> <compiler options>
```

Example

```
$ icc  foo.c  –o  foo.out

$ icpc foo1.cpp –o foo1.out
```

icc	Intel® C Compiler
icpc	Intel® C++ Compiler
ifort	Intel® fortran Compiler
-openmp	Link Intel® version of Openmp
-mkl	Link Intel® Math Kernel Library
-tbb	Link Intel® Thread Building Blocks
-O0	Compile program with no Optimizations
-O1/-O2/-O3	Compile program with different levels of optimization
-opt-report=5	Generate Optimization Report of level 5 (MAX 5)
-mmic	Generates executable for MIC architecture (Intel® Xeonphi Arch)
-xHOST	Generates optimized executable with Highest possible instruction set with current processor
-ipo	Enables inter procedural optimizations between multiple source files
-guide-vec[=n]	Guidance for auto vectorization. n is level of auto vectorization from 1 to 4
-g	Generate Symbolic debugging information in the object file

OpenMP

Compile OpenMP Program:

```
$icc <C filename> -o <executable filename> -openmp
```

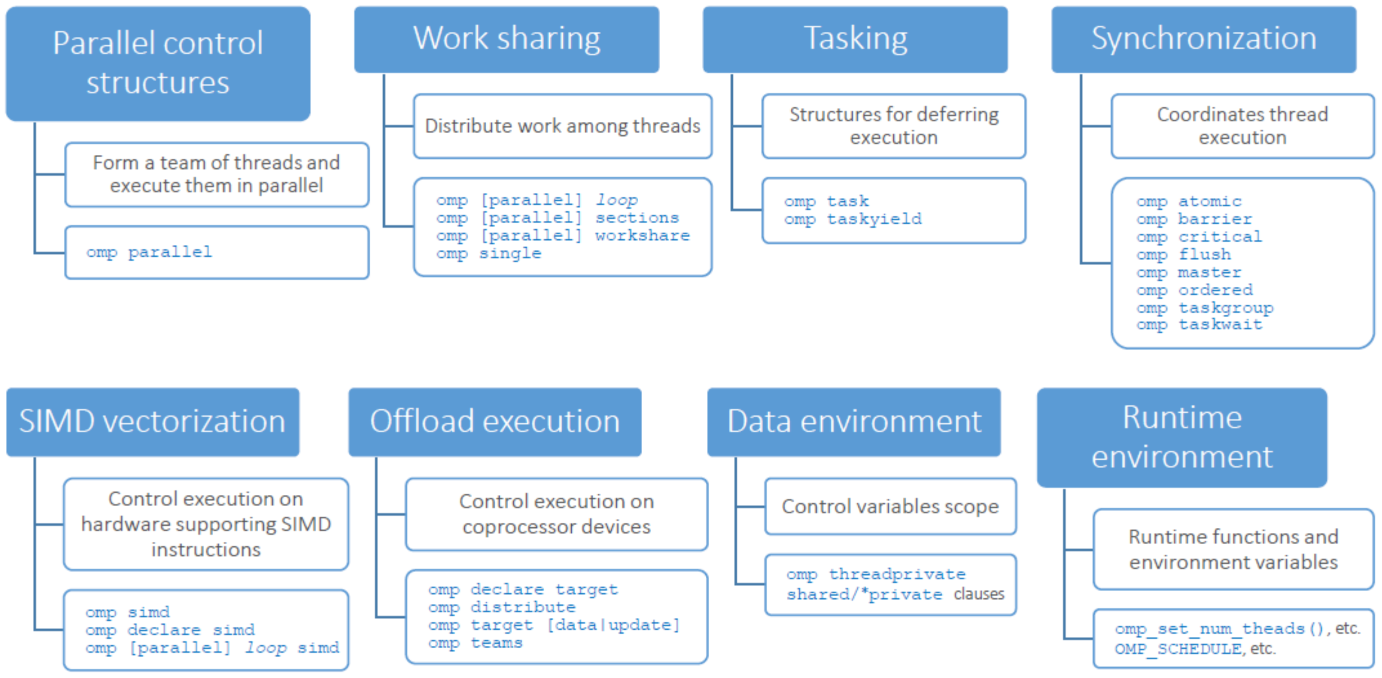
Example

```
$icc  foo.c -o foo.out -openmp
```

Structure

```
#pragma omp parallel <directives>
```

Core Elements:



OpenMP Directives:

parallel	Defines a parallel region to be executed by multiple threads.
for	Iterative work sharing construct. Iterations of the *for* loop are distributed across spawned threads.
sections	Non-iterative work-sharing construct that specifies a set of constructs that are to be divided among threads in a team and executed once.
single	Associated structured block is executed by only one thread in the team (not necessarily the master thread).

parallel for	A shortcut form for a parallel region that contains only *for* directive.
master	Associated structured block is executed by the master thread of the team.
barrier	Used to synchronize. Each thread waits for other threads to reach this point. After all threads encountering the barrier, each thread begins executing further statements in parallel.
atomic	Specifies memory location that must be updated sequentially. One thread at a time.

OpenMP Library Routines:

omp_set_num_threads(int)	Sets the maximum number of threads at runtime
int omp_get_num_threads( )	Returns total number of active threads
int omp_get_thread_num( )	Returns thread number
num_threads(int)	Sets the total number of threads to spawn
double omp_get_wtime( )	Returns current time of the machine
int omp_num_procs( )	Returns number of logical processors
int omp_get_max_threads( )	Returns maximum available threads
omp_set_dynamic(0/1)	Enable (1) or disable(0) the dynamic adjustment of the number of threads within a team
omp_set_nested(0/1)	Enable(1) or disable(0) nested parallelism

OpenMP Environment Variables:

OMP_NUM_THREADS	Specifies the default number of threads to use in parallel regions.
OMP_NESTED	Enable or disable nested parallel regions.
OMP_DYNAMIC	Enable or disable the dynamic adjustment of the number of threads within a team
OMP_WAIT_POLICY	Specifies whether waiting threads should be active or passive.
OMP_MAX_ACTIVE_LEVELS	Specifies the initial value for the maximum number of nested parallel regions.

OpenMP 4.0 Features:

Type	Name	Description
Directives	simd	Indicates loop to be transformed for SIMD vectorization
	loop simd	Specifies that a loop that can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel by threads in the team
	target[data]	Creates a device data environment for the extent of the region and also starts executing on the device
	target update	Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.
	teams	Creates a league of thread teams where the master thread of each team executes the region.
	distribute[simd]	Distribute specifies loops which are executed by the thread teams. Distribute simd specifies loops which are executed concurrently using SIMD instructions
	cancel	Requests cancellation of the innermost enclosing region of the type specified. The cancel directive may not be used in place of the statement following an if, while, do, switch, or label.
	cancellation point	Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been requested
	declare reduction	Allows user to declare a reduction-identifier for user defined reduction operation
Library Routines	omp_get_proc_bind	Returns the thread affinity policy to be used for the subsequent parallel regions(includes nested) that do not specify a proc_bind clause
	omp_set_default_device	Controls the default target device by assigning the value of the default-device-var ICV
	omp_is_initial_device	Returns true if the current task is executing on the host device; otherwise, it returns false.
Environment Variables	OMP_CANCELLATION	Sets the cancel-var ICV. policy may be true or false. If true, the effects of the cancel construct and of cancellation points are enabled and cancellation is activated
	OMP_DEFAULT_DEVICE	Sets the default-device-var ICV that controls the default device number to use in device constructs
	OMP_DISPLAY_ENV	If var is TRUE, instructs the runtime to display the OpenMP version number and the value of the ICVs associated with the environment variables as name=value pairs. If var is VERBOSE , the runtime may also display vendor-specific variables. If var is FALSE, no information is displayed
	OMP_PLACES	Sets the place-partition-var ICV that defines the OpenMP places available to the execution environment. places is an abstract name (threads, cores, sockets, or implementation-defined), or a list of non-negative numbers

OpenMP4.0 features reference: http://openmp.org/mp-documents/OpenMP-4.0-C.pdf

Intel® MPI (Message Passing Interface)

Important terms:

RANK	Each Process is assigned an identifier (contiguous integers starting from zero) called rank.	
GROUP	Group is an ordered set of processes. There are many predefined groups. (MPI_GROUP_EMPTY, MPI_GROUP_NULL)	
	MPI_Group_incl(MPI_Group group, int n, const int ranks[ ], MPI_Group *newgroup)	Produces a group by reordering an existing group and taking only listed members.
	MPI_Group_size(MPI_Group group, int *size)	Returns the size of a group
	MPI_Group_free(MPI_Group *group)	Frees a group
COMMUNICATOR	Mechanism through which scope of process communication is determined. It is a dynamic object that is created, used and destroyed. MPI_COMM_WORLD is a universal inter-communicator for all processes, available immediately after MPI_init()	
	MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)	Creates a new communicator
	MPI_Comm_get_attr(MPI_Comm comm, int comm_keyval, void *attribute_val, int *flag)	Retrieves attribute value by key
	MPI_Comm_join(int fd, MPI_Comm *intercomm)	Establishes communication between MPI jobs
	MPI_Comm_free(MPI_Comm *comm)	Mark a communicator object for deallocation

MPI Datatypes:

MPI Datatypes	**C Datatypes**
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Set Compiler path and compiler related environment variables

```
      $source <install-dir>/bin64/mpivars.sh
```

Invoke MPI C compiler

```
      $ mpiicc  <C filename >  -o  <Executable filename> <compiler options>
```

Executing an MPI executable

```
      $ mpirun –np <int> -machinefile <File with the list of IP's> <Executable
filename>
```

Example

Compile:

```
      $ mpiicc  foo.c  –o  foo.out
```

Execute:

```
      $ mpirun –np 15 –machinefile machinefile ./foo.out
```

MPI Library Routines:

MPI_Init (&argc, &argv);	Intialize MPI
MPI_Comm_size (MPI_COMM_WORLD, &size);	Number of process in MPI_COMM_WORLD
MPI_Comm_rank (MPI_COMM_WORLD, &rank);	rank of each process in MPI_COMM_WORLD
MPI_Bcast(void *data, int length, MPI_Datatype, source, MPI_Comm communicator);	Broadcast data of size length and type MPI_Datatype from source to all the process in communicator
MPI_Send(void* data, int count, MPI_Datatype datatype, int destination,int tag, MPI_Comm communicator)	Send data of size count and type datatype to destination(rank) in communicator with tag
MPI_Recv(void* data, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm communicator, MPI_Status* status)	Receive data of size count and type datatype from source(rank) in communicator with tag
MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)	Begins Non-blocking Send

MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)	Begins Non-blocking Send
MPI_Put(const void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)	Put data into a memory window on a remote process
MPI_Get(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)	Get data from a memory window on a remote process
int MPI_Barrier(MPI_Comm comm)	Blocks until all processes in the communicator have reached this routine.
MPI_Finalize(void)	Terminates MPI execution environment

Intel® MPI Environment Variables:

I_MPI_DEBUG	Print out debugging information when an MPI program starts running
I_MPI_DEBUG_OUTPUT	Set output file name for debug information.
I_MPI_PERHOST	Define the default settings for the -perhost option in the mpiexec command
I_MPI_JOB_TIMEOUT	Set the mpiexec/mpirun timeout
I_MPI_OUTPUT_CHUNK_SIZE	Set the size of the stdout/stderr output buffer
I_MPI_PMI_EXTENSIONS	Turn on/off the use of the Intel® MPI Library Process Management Interface (PMI) extensions.
I_MPI_PLATFORM	Select the intended optimization platform.
I_MPI_PLATFORM_CHECK	Turn on/off the optimization setting similarity check.

MPI Derived Datatypes (User Defined Datatypes):

Other than the predefined MPI datatypes, it is possible to define new datatypes by grouping. This class of data is the derived datatype. Derived datatypes in MPI can be used in:

- Grouping data of different datatypes for communication
- Grouping noncontiguous data for communication

Library Routine	**Description**
MPI_Type_indexed(int count, const int array_of_blocklengths[ ], const int array_of_displacements[ ], MPI_Datatype oldtype, MPI_Datatype *newtype)	Creates an indexed datatype
MPI_Type_commit(MPI_Datatype *datatype)	Commits a data type
MPI_Type_free(MPI_Datatype *datatype)	Frees a data type

MPI Error handling:

MPI provides user reliable message transmission, thus doesn't provide mechanism to deal with communication system. MPI returns an error code when an error is encountered. By default, all MPI errors aborts the parallel computation. The desired behavior is that a relevant error code be returned, and the effect of the error be localized to the greatest possible extent.

Library Routines	**Description**
int MPI_Error_class(int errorcode, int *errorclass)	Converts an error code into an error class
int MPI_Error_string(int errorcode, char *string, int *resultlen)	Returns a string for a given error code
int MPI_Errhandler_free (MPI_Errhandler *errhandler)	Frees an MPI-style error handler

Error Code	**Description**
MPI_SUCCESS	No error
MPI_ERR_BUFFER	Invalid buffer pointer
MPI_ERR_COUNT	Invalid count argument
MPI_ERR_TYPE	Invalid datatype argument
MPI_ERR_TAG	Invalid tag argument
MPI_ERR_COMM	Invalid communicator
MPI_ERR_RANK	Invalid rank
MPI_ERR_REQUEST	Invalid request
MPI_ERR_ROOT	Invalid root

MPI_ERR_GROUP	Invalid Group
MPI_ERR_OP	Invalid operation
MPI_ERR_TOPOLOGY	Invalid topology
MPI_ERR_DIMS	Invalid dimension argument
MPI_ERR_ARG	Invalid argument of some other kind
MPI_ERR_UNKNOWN	Unknown error
MPI_ERR_TRUNCATE	Message truncated on receive
MPI_ERR_OTHER	Known error not in the list
MPI_ERR_IN_STATUS	Error code in status
MPI_ERR_PENDING	Pending request
MPI_ERR_LASTCODE	Last code error

Intel® Threading Advisor

Threading Advisor is part of Intel® Advisor XE 2016. It's a threading design and prototyping tool that lets you analyze, design, tune and check threading design options without disrupting your normal development.

Set Tool path and Tool related environment variables

```
    $source <install-dir>/advisor_xe/advixe-vars.sh
```

Prerequisites:

To build applications that produce the most accurate and complete Threading Advisor analysis results, build an optimized binary of your application in release mode using these compiler/linker settings:

```
    $icc <C filename> -g  -I${ADVISOR_XE_2016_DIR}/include –O2 –ldl –Bdynamic
-o <Executable filename>
```

Example:

```
    $icc foo.c –g -I${ADVISOR_XE_2016_DIR}/include –O2 –ldl –Bdynamic –o foo.out
```

Note:

- Verify your application runs, before trying to analyze it with the Intel Advisor.
- Make sure you run the Intel Advisor in the same environment as your application.

Invoke Intel® Advisor XE GUI:

```
    $advixe-gui
```

Invoke Intel® Advisor XE CLI:

```
    $ advixe-cl -collect survey –project-dir ./<Executable Filename> -- <Project
Name>
```

Intel® VTune Amplifier XE

Intel® VTune™ Amplifier XE is a Performance profiler targeted for analysis of applications running on local and remote Linux systems. Use this tool to analyze the algorithm choices, find serial and parallel code bottlenecks, understand where and how your application can benefit from available hardware resources, and speed up the execution.

Set Tool path and Tool related environment variables

```
$source <install-dir>/ vtune_amplifier_xe_2016/ amplxe-vars.sh
```

Prerequisites:

To build applications that produce the most accurate and complete VTune Amplifier profiling results, build an optimized binary of your application in release mode using these compiler/linker settings:

```
$icc <C filename> –g  –O2 –debug inline-debug-info  -o <Executable filename>
```

Example:

```
$icc foo.c –g –O2 –debug inline-debug-info –o foo.out
```

Note:

- Verify your application runs before trying to analyze it with the Intel ® Amplifier XE.
- Make sure you run the Intel ® Amplifier XE in the same environment as your application.

Invoke Intel® VTune Amplifier XE GUI:

```
$amplxe-gui
```

Invoke Intel® VTune Amplifier XE CLI:

```
$ amplxe-cl -collect <analysis_type> [--] <target>
```

Intel® Trace Analyzer and Collector (ITAC)

Intel® Trace Analyzer and Collector enables you to understand MPI application behavior, quickly find bottlenecks and achieve high performance for parallel cluster applications.

Use this tool to do the following:

- Evaluate profiling statistics and load balancing
- Learn about communication patterns, parameters, and performance data
- Identify communication hotspots
- Decrease execution time and increase application efficiency

Set Tool path and Tool related environment variables

```
$source <install-dir>/ itac/<version number>/bin/ itac-vars.sh
```

Prerequisites:

Execute an optimized binary of your MPI application using this option to generate *.stf file:

```
$mpirun –np <int> <Executable filename> -trace
```

Example:

```
      $mpirun –np 5 ./foo.out -trace
```

Invoke Intel® TAC GUI:

```
      $traceanalyzer
```

Invoke Intel® TAC GUI along with tracefile:

```
      $traceanalyzer foo.stf
```

Vectorization

Ways To Vectorize:

- Intel® Compilers Auto-Vectorization
- Intel® Compilers Auto-Vectorization with (Compiler Hints like #pragma)
- OpenMP SIMD Vectorization
- Vector Intrinsic and Array notations

Intel® Compiler Options for vectorization

Option	**Description**
-O2	Enables intra-file inter procedural optimizations for speed, including: • Vectorization • Loop unrolling
-03	Performs O2 optimizations and enables more aggressive loop transformations such as: • Loop fusion • Block unroll-and-jam • Collapsing IF statements This option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets. However, it might incur in slower code, numerical stability issues, and compilation time increase.
-xHost	Tells the compiler which processor features it may target, referring to which instruction sets and optimizations it may generate (not available for Intel® Xeon Phi architecture).
-[no-]vec	enables(DEFAULT)/disables vectorization
-vec-threshold[n]	sets a threshold for the vectorization of loops based on the probability of profitable execution of the vectorized loop in parallel
-[no-]simd	enables(DEFAULT)/disables vectorization using simd pragma
-ansi-alias	option allows the compiler to assume strict adherence to the aliasing rules in the ISO C standard. Use these options responsibly; if you use these options when memory is aliased it may lead to incorrect results (Disambiguation of pointers and arrays)
-opt-report=[n]	(n<=5)indicate vectorized loops(DEFAULT when enabled)
-opt-report-phase=vec	Indicates only vectorized loops in optimization reports

Intel® Specific Compiler Directives

#pragma	**Description**
simd	Enforce vectorization ; ignore dependencies
ivdep	Place before a loop to control vectorizaton/software pipelining The compiler is instructed to ignore "assumed" (not proven) dependencies preventing vectorization/software pipelining. For Itanium: Assume no BACKWARD dependencies, FORWARD loopcarried dependencies still can exist w/o preventing SWP. Use with –ivdep_parallel option to exclude loopcarried dependencies completely (e.g. for indirect addressing)
vector always	Always vectorize
vector aligned	use aligned load/store instructions
vector unaligned	use unaligned load/store instructions
loop count(n)	Place before a loop to communicate the approximate number of iterations the loop will execute. Affects software pipelining, vectorization and other loop transformations.
distribute point	Placed before a loop, the compiler will attempt to distribute the loop based on its internal heuristic. Placed within a loop, the compiler will attempt to distribute the loop at the point of the pragma. All loop-carried dependencies will be ignored
nontemporal	directs the compiler to use nontemporal (that is, streaming) stores on systems based on all supported architectures, unless otherwise specified; optionally takes a comma separated list of variables
temporal	directs the compiler to use temporal (that is, non-streaming) stores on systems based on all supported architectures, unless otherwise specified
unroll, nounroll, unroll(n)	Place before an inner loop (ignored on non-inmost loops). #pragma unroll without a count allows the compiler to determine the unroll factor. #pragma unroll(n) tell the compiler to unroll the loop n times. #pragma nounroll is the same as #pragma unroll(0).
vecremainder	instructs the compiler to vectorize the remainder loop when the original loop is vectorized
novecremainder	instructs the compiler not to vectorize the remainder loop when the original loop is vectorized

OpenMP SIMD Directive clauses

Clauses	Description
safelen(n)	Safelen clause is used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than its value
collapse(n)	Collapse clause may be used to specify how many loops are associated with the construct.
aligned(list[:linearstep])	Aligned clause declares that the object to which each list item points is aligned to C/C++ the number of bytes expressed in the optional parameter of the aligned clause
private(list)	Specifies that each thread should have its own instance of a variable.
lastprivate(list)	Specifies thread that executes the ending loop index copies its value to the master (serial) thread this gives the same result as serial execution
reduction(operator:var1, var2,...,varN)	Loop code implements reduction (like "+") on arguments listed which can be vectorized
linear(list[:linear-step])	Linear clause declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop

Intel® Vector Advisor

Vector Advisor is part of Intel® Advisor XE 2016. It's a is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization, identify what is blocking effective vectorization, explore the benefit of alternative data reorganizations, and increase the confidence that vectorization is safe

Set Tool path and Tool related environment variables
```
$source <install-dir>/advisor_xe/advixe-vars.sh
```

Prerequisites:
To build applications that produce the most accurate and complete Threading Advisor analysis results, build an optimized binary of your application in release mode using these compiler/linker settings:
```
$icc <C filename> –g –O2/O3 –opt-report=5   -o <Executable filename>
```

Example:
```
$icc foo.c –g –O2 –opt-report=5 –o foo.out
```

Note:

- Verify your application runs before trying to analyze it with the Intel Advisor.
- Make sure you run the Intel Advisor in the same environment as your application.

Invoke Intel® Advisor XE GUI:
```
$advixe-gui
```

Invoke Intel® Advisor XE CLI:

```
$ advixe-cl -collect survey –project-dir ./<Executable Filename> -- <Project
Name>
```