# Optimization of Iso3DFD with Intel® Advisor, Intel® VTune™ Amplifier and Intel® C++ & Fortran Compiler (Linux*)

---

# Lab



Developer Product Division
Software and Service Group

# Disclaimer

The information contained in this document is provided for informational purposes only and represents the current view of Intel Corporation ("Intel") and its contributors ("Contributors") on, as of the date of publication. Intel and the Contributors make no commitment to update the information contained in this document, and Intel reserves the right to make changes at any time, without notice.

DISCLAIMER. THIS DOCUMENT, IS PROVIDED "AS IS." NEITHER INTEL, NOR THE CONTRIBUTORS MAKE ANY REPRESENTATIONS OF ANY KIND WITH RESPECT TO PRODUCTS REFERENCED HEREIN, WHETHER SUCH PRODUCTS ARE THOSE OF INTEL, THE CONTRIBUTORS, OR THIRD PARTIES. INTEL, AND ITS CONTRIBUTORS EXPRESSLY DISCLAIM ANY AND ALL WARRANTIES, IMPLIED OR EXPRESS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, NON-INFRINGEMENT, AND ANY WARRANTY ARISING OUT OF THE INFORMATION CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION, ANY PRODUCTS, SPECIFICATIONS, OR OTHER MATERIALS REFERENCED HEREIN. INTEL, AND ITS CONTRIBUTORS DO NOT WARRANT THAT THIS DOCUMENT IS FREE FROM ERRORS, OR THAT ANY PRODUCTS OR OTHER TECHNOLOGY DEVELOPED IN CONFORMANCE WITH THIS DOCUMENT WILL PERFORM IN THE INTENDED MANNER, OR WILL BE FREE FROM INFRINGEMENT OF THIRD PARTY PROPRIETARY RIGHTS, AND INTEL, AND ITS CONTRIBUTORS DISCLAIM ALL LIABILITY THEREFOR. INTEL, AND ITS CONTRIBUTORS DO NOT WARRANT THAT ANY PRODUCT REFERENCED HEREIN OR ANY PRODUCT OR TECHNOLOGY DEVELOPED IN RELIANCE UPON THIS DOCUMENT, IN WHOLE OR IN PART, WILL BE SUFFICIENT, ACCURATE, RELIABLE, COMPLETE, FREE FROM DEFECTS OR SAFE FOR ITS INTENDED PURPOSE, AND HEREBY DISCLAIM ALL LIABILITIES THEREFOR. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT OR TECHNOLOGY DOES SO AT HIS OR HER OWN RISK.

Licenses may be required. Intel, its contributors and others may have patents or pending patent applications, trademarks, copyrights or other intellectual proprietary rights covering subject matter contained or described in this document. No license, express, implied, by estoppels or otherwise, to any intellectual property rights of Intel or any other party is granted herein. It is your responsibility to seek licenses for such intellectual property rights from Intel and others where appropriate. Limited License Grant. Intel hereby grants you a limited copyright license to copy this document for your use and internal distribution only. You may not distribute this document externally, in whole or in part, to any other person or entity. LIMITED LIABILITY. IN NO EVENT SHALL INTEL, OR ITS CONTRIBUTORS HAVE ANY LIABILITY TO YOU OR TO ANY OTHER THIRD PARTY, FOR ANY LOST PROFITS, LOST DATA, LOSS OF USE OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF THIS DOCUMENT OR RELIANCE UPON THE INFORMATION CONTAINED HEREIN, UNDER ANY CAUSE OF ACTION OR THEORY OF LIABILITY, AND IRRESPECTIVE OF WHETHER INTEL, OR ANY CONTRIBUTOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

See: http://software.intel.com/en-us/articles/optimization-notice/

| | |
|---|---|
| **Requirements** | **Intel® Parallel Studio XE 2018 Composer Edition** with **Intel® C++/Fortran Compiler**<br><br>**Linux*** host OS supported by Intel® C++/Fortran Compiler<br><br>At least **2nd generation Intel® Core™ processor** (with Intel® AVX).<br>For stable measurements it is recommended to turn off Intel® Hyper-Threading, Intel SpeedStep® and Intel® Turbo Boost if available. |
| **Objective** | In this lab session, you will use the Intel® C++/Fortran Compiler as well as Intel® Advisor and Intel® VTune Amplifier to become familiar with vectorization, threading, and optimization.<br>After successfully completing this lab's activities, you will be able to:<br>• To select the right instruction set extension switch for compiling<br>• Apply and analyze vectorization<br>• Use OMP for threading for loops<br>• Improve vectorization by using the OMP directives<br>• Understand OpenMP* 4.0 techniques for vectorization<br>• Understand Intel specific techniques for vectorization<br>• Understand how to run the roofline model in Intel® Advisor<br>• Understand how to compare 2 Roofline Models in Advisor<br>• Run the Memory Access Pattern analysis and the Dependency analysis in Intel® Advisor<br>• Run the cache simulation and extract information in Intel® Advisor<br>• Run the HPC Performance Characterization and the Memmory Access analysis in VTune Amplifier |

**Notes**:

•

# Preface

The following activities can be used for 2nd generation Intel® Core™ processor and later (with Intel® AVX/AVX-512 support, Intel® Xeon Phi™ processor). Please regard the following for the different targets:

- **2nd generation Intel® Core™ processor and later:**
  - As **`$SIMD`** use **`–xavx`** (alternatively you can also use **`–mavx`** or **`–axavx`**)
  - Please keep in mind: An AVX **vector contains 4 double precision FP elements** or 8 single precision FP elements. The activities below use double precision FP if not noted otherwise, but can be also changed to single precision FP.
- **6th generation Intel® Core™ processor (SKX):**
  - As **`$SIMD`** use **`–xCORE-AVX512`**
  - Please keep in mind: An AVX-512 **vector contains 8 double precision FP elements** or 16 single precision FP elements if 512 bit zmm registers are used. By default the compiler avoid zmm code generation with 18.0 compiler version for SKX with –xCORE-AVX512 option and generate AVX-512 instructions with 256 bit ymm registers in most cases. Use -qopt-zmm-usage=high to force the compiler generate zmm code.
- **Intel® Xeon Phi™ processor (KNL):**
  - As **`$SIMD`** use **`–xMIC-AVX512`**
  - An AVX-512 **vector contains 8 double precision FP elements** or 16 single precision FP elements if zmm registeres are used. By default the compiler generate zmm code with 18.0 compiler version for KNL with –xMIC-AVX512 option.

**Important note:**
Please don't forget to source the **`compilervars.[csh|sh]`** script to get access to the compiler and libraries. All exercises require that if not noted otherwise below or by the instructor. Depending on the installation of the compilers, issue the following command:

```
$ source <compiler_root>/bin/compilervars.[csh|sh] intel64
```
(64 bit compiler)

or

```
$ source <compiler_root>/bin/compilervars.[csh|sh] ia32
```
(32 bit compiler)

# C/C++

## Activity 1.1 – Characterization and first profiling of iso3DFD (15 mn)

In this activity, we will first run the unoptimized version of iso3DFD

1. Navigate to the **`iso3dfdlab/`** folder. This is the root directory of the iso3DFD project. The sources are located in **`iso3dfdlab/src.`** To keep tracks of every optimization, we will create a new folder for each version. For creating a new version, you can use the script **`iso3dfdlab/createnewversion.pl.`**
2. Initially, the script generates a binary for AVX. Modify the script to target your architecture. Open **`iso3dfdlab/createnewversion.pl`** and change the line:
   `my $IS="avx";`
   Modify "avx" to reflect your instruction set:
   a. For Sandy Bridge and Ivy Bridge: "AVX"
   b. For Haswell and Broadwell: "CORE-AVX2"
   c. For KNL: "MIC-AVX512"
   d. For Skylake: "CORE-AVX512"
   e. For both KNL and Skylake: "COMMON-AVX512"
3. Once the script has been modified. You can run it.
   **`$iso3dfdlab/createnewversion.pl`**
   It will generate 2 additional scripts
   a. compileversion01.sh: You can run this script to compile the new version
   b. runversion01.sh: You can run this script to execute the new version
4. Now you can compile and tun the dev01 by running:
   **`$iso3dfdlab/compileversion01.sh`**
   **`$iso3dfdlab/runversion01.sh`**
   Record the performance in GFlops/s
   GFlop/s _____.
5. Before modifying the code, we need to characterize it first. Intel® Advisor comes with a powerful characterization tool called the Roofline Model. The Roofline Model offers an easy way to understand if a loop or a function is computation intensive or memory intensive. Characterizing an application is a mandatory step before doing any optimization (Otherwise you may try to optimize a loop that is already running fine)
   Open Advisor and create a new project.

   **`$ advixe-gui`**

   Set the project name to "Iso3DFD dev01". Then we need to setup the project.
   a. Application: you must setup the iso3DFD binary. It must point to **$iso3dfdlab/bin/iso3dfd_dev01_cpu_[YOUR_INSTRUCTION_SET].exe**.
   b. Application parameters: In this case, we will provide additional command line parameters to replicate the parameters used in the script **runversion01.sh**. You need to set **"128 128 128"** in this field.

You can validate. Now on the left hand side, you should see "**Run Roofline**". Click on it.

6. The Roofline model consists in 2 runs of your application.
    a. The first one use sampling with a very low overhead in order to time your application.
    b. The second one uses instrumentation with higher overhead to count flops and memory movements.
7. Note that Intel® Advisor can also be run in command line. For the Roofline model, you can run:
   **$advixe-cl -c roofline -project-dir advi01 --search-dir src:r=./src -- bin/iso3dfd_dev01_cpu_avx.exe**
   On KNL, you need to run the 2 analyses separately:
   **$advixe-cl -c survey -project-dir advi01 --search-dir src:r=./src -- bin/iso3dfd_dev01_cpu_avx.exe**
   **$advixe-cl -c tripcounts -flop -project-dir advi01 --search-dir src:r=./src -- bin/iso3dfd_dev01_cpu_avx.exe**

8. Now look at the results in the roofline model. A quick look shows that we are under the DRAM roofline. We might have some room for improvement. In order to optimize we are going to check each of the following key points:
    a. Threading
    b. Memory accesses
    c. Vectorization

   You can already have a look at Intel® Advisor recommendations by switching to the "survey" tab. We will come back to these recommendations later.

# Activity 1.2 – Dev01 Optimizing the Threading (10mn)

Our first concern will be to verify the state of the threading in this application. Threading must be considered in any HPC application as modern CPUs offer a large number of cores and threads. Taking advantage of this computation power can significantly increase the performance of your applications.

1. We are first going to analyze how performant is the threading in our application. We've seen with the Roofline Model that some improvement seems possible. The next step is to use Intel® VTune Amplifier to get metrics about the threading (but also memory).
   VTune comes with an analysis called "HPC Performance Characterization" that perfectly suits our need.
   Open VTune, setup your project the same way you did for Advisor in the previous activity.
2. Once you have defined a binary and the command line parameters (128 128 128) you can validate and select the "HPC Performance Characterization" analysis.
   Run it and look at the results. What do you observe regarding the threading ?
   _____
   _____
   _____

3. We are going to use OpenMP to add the threading here. Try to find in the file **`iso3dfdlab/src/dev01/iso-3dfd_parallel.cc`** the appropriate location for a **#pragma omp parallel for** clause. Keep in mind that for reducing the overhead, we should always try to generate the parallelism at the highest level possible.
   When you are ready, add the OMP clause on the top of the appropriate for loop.

4. Once you have modified the code, run again the dev01
   **`$iso3dfdlab/runversion01.sh`**
   Record the performance in GFlops/s
   GFlop/s _____.

5. Create a new code version using the script createnewversion.pl
   $ **`iso3dfdlab/createnewversion.pl`**

# Activity 1.3 – Dev02 Memory Optimization (20mn)

Previously we have been able to identify that some improvement was possible in our code using the Roofline Model. Then running a VTune analysis it showed that no threading was implemented. Now that this step has been completed, we can move forward and continue our optimization work. In this activity, we will check the memory access pattern. In HPC application, it is important to understand how data are accessed. To reach optimal performance, it is always preferable to access data using unit stride accesses. Accessing the data with constant strides or unpredictable access often results in reduced performance.

1. We are going to create a new project in Advisor (You can call it "Iso3DFD dev02")
2. Run a new Roofline Model to verify that some improvement is still possible
3. You can use the compare feature of the Roofline Model to compare the performance your initial dev02 with the run you did on dev01 (before implementing the threading) Comparing 2 rooflines is sometime useful to understand where performance benefit come from between 2 code versions.
4. After running the Roofline Model, do you thing that some improvement is still possible ?
5. Now we want to verify if we have any memory related limitations. For this, you can run 3 different analyses
    a. In VTune, the HPC Performance Characterization will give you a very understandable summary of your memory limitations
    b. In VTune, the memory access analysis will provide more advanced details regarding memory utilization (Bandwidth in real time, cache misses, etc)
    c. In Intel® Advisor, using the "Memory Access Pattern" analysis, you can identify inefficient memory access patterns.

    A good methodology is to check first with VTune that a bottleneck exists regarding the memory utilization (use the HPC Performance Characterization) and then if VTune reports some memory problems, you can use Advisor to identify the reason of this bottleneck.
6. Start running a new HPC Performance Characterization in VTune and verify if a memory problem seems to exist
7. Run the Memory Access Pattern (MAP) in Intel® Advisor.
    a. Open your "Iso3DFD dev02" project in Advisor (You've already run the roofline which consists in survey and flops)
    b. Open the project parameters  and modify the command line parameters to 64 64 64. This step must be done otherwise the MAP analysis will take too much time.
    c. Click on the "survey" tab
    d. Check the checkbox of the 3 or 4 main hotspots
    e. Then run the Memory Access Pattern analysis
8. Analyze the results of the MAP analysis. Do you see non unit strides ? Try to identify which accesses are responsible in the poor performance

_____

_____

9. Modify the code to resolve the wrong memory access pattern (ask for help if you don’t see what to do)
10. Run the dev02 again and write the performance:

    GFlop/s _____.

At this step, you are able to run the Roofline Model, identify an application that can benefit from code improvement. You also know how to identify some problems due to data accesses.

# Activity 1.4 – Dev03 Cache blocking (30mn)

In this activity, we will spend more time on memory optimizations. Many HPC applications are limited by bandwidth because applications arithmetic intensity is usually lower than the platform ideal arithmetic intensity. Therefore, being able to reuse as much as possible the data already loaded into the cache is crucial. Here, we will try to identify cache related problem and we will implement a technic known as Cache Blocking that can help improving data re-usage.

1. Create a new version of the code using the script createnewversion.pl
   $ **iso3dfdlab/createnewversion.pl**
2. Again create a new project in Advisor (iso3dfdlab dev03) and run the roofline model (You should know how to do it now). Verify that performance improvement still looks possible and compare this new result with the previous one using the compare feature.
3. Some improvement is still doable and we are going to analyze the memory with VTune
4. Open VTune, create a new analysis for dev03 and run a "Memory Access" analysis. Try to understand the results regarding memory utilization and cache misses. At this stage, you should be able to see a significant number of cache misses coming from the 2 last levels of cache (and most of them should come from the LLC). This behavior might be optimized to move some of the LLC misses to L2.
5. The next step is to implement cache blocking. Cache blocking is a technic that changes the iteration process to create smaller 2D or 3D blocks. Creating these small blocks usually offers a higher cache re-usage. A general methodology in 2D is to process as follow:
   If the initial loops looks like this

```
for(int i=0;i<N;i++){
  for(int j=0;j<M;j++){
    //do something
  }
}
```

   A cache blocking implementation would look like this:

```
for(cbi=0;cbi<N;cbi+=SIZE_BLOCK_I){
for(cbj=0;cbj<M;cbj+=SIZE_BLOCK_J){
  for(int i=cbi;i<min(cbi+SIZE_BLOCK_I, N);i++){
    for(int j=cbj;j< min(cbj+SIZE_BLOCK_J, M);j++){
      //do something
    }
  }
}}
```

   In Iso3DFD, the cache blocking can be implemented on the loop following the **#pragma omp parallel for** clause.
   Once the implementation is done, test it. You might have a performance degradation based on the size you provide for you blocks.
   What usually works well is to avoid blocking in the unit stride dimension due to prefetching. Then you want to set up a size that fits into the L2 when you multiply your 3 dimensions.
   Ask the trainer if you need more details on cache blocking implementation.
6. Once you have played with the cache blocking parameters and that the performance gets better, keep track of it and go switch to the next activity:

   GFlop/s _____.

Bonus: It is also possible to use the cache simulation in Intel® Advisor to test the efficiency of your cache blocking. This feature is still a preview feature

1. Close Advisor (You must close every instance of it)
2. In the terminal, run:
   **$export ADVIXE_EXPERIMENTAL=cachesim**
3. Open Advisor by running:
   **$advixe-gui**
4. In the project setting, go to Memory Access Pattern. At the bottom, you can enable the cache simulation feature. Then you need to setup the cache. To replicate your system (it might not be possible yet on all configuratins), you can read your cache settings by doing:

   **$cat /sys/devices/system/cpu/cpu[X]/cache/index[Y]/***
   If you want more details, ask the trainer

# Activity 1.5 – Dev04 Enabling vectorization (15mn)

At this stage, we've already solved many problems related to threading and memory utilization. In this activity, we will focus on vectorization. Vectorization is the ability to use the Single Instruction Multiple Data feature (SIMD) from the processor. Vectorization can bring an important performance speedup especially on AVX512 architectures.

1. As usual, create a new version using the script createnewversion.pl
   $ **`iso3dfdlab/createnewversion.pl`**
2. Again create a new project in Advisor (iso3dfdlab dev04) and run the roofline. Verify that performance improvement still looks possible and compare this new result with the previous one using the compare feature.
3. Once you have run the Roofline Model and verified that some improvement seems possible, click on the "survey" tab in Advisor and look at the first loops. It seems that Advisor detects some potential vectorization. In this case, the compiler was not able to use auto-vectorization due to potential conflict in the variables.
4. Look at the recommendations, Advisors recommend to run a dependency analysis on few loops to check if the dependency is real.
   Select the loops accordingly to Advisor's recommendations and run a dependency analysis (Click on collect on the left hand side under "Check Dependencies")
5. Doing this analysis might take some time. Look at the results when they are available.
   What does Advisor tells you regarding vectorization of the outer loop ? Is it safe ?
   Note: Always be careful when Advisor tells you that vectorization is safe. Dependency analysis only relies on what Advisor observed during the run. Changing your application parameters or inputs might change how the application behave. Be sure to know what you are doing when you force vectorization.
6. As vectorization is safe here, we need to help the compiler to let it know that compilation is safe. Find the good location for adding a **#pragma omp simd** clause in the code
7. Compile and run the code again to get the new performance:

   GFlop/s _____.

# Activity 1.6 – Dev05 Detecting the NUMA effect (30mn)

In this activity, we are going to check how NUMA effect can affect the performance of your application. This activity only applies to multi sockets nodes. If you are running on KNL, you might want to skip it unless your KNL is in Sub-NUMA clustering mode.

1. As usual, create a new version using the script createnewversion.pl
   $ **`iso3dfdlab/createnewversion.pl`**
2. Compile the new version and open VTune Amplifier. Setup a new project and run the memory access analysis. Once the analysis has been ran, check if you see some activity on the QPI. If you see a significant activity on the QPI, it means that data are traveling from one NUMA domain to another one. This is usually something we want to avoid as the latency od accessing the data on a remote NUMA node is much higher than getting it from the local memory.
3. Another efficient way to check that your application is sensitive to NUMA effect is to use NUMACTL
   $ echo "numactl --membind=0 --cpunodebind=0 ./myapp && numactl --membind=1 --cpunodebind=1 ./myapp" > script.sh
   $ time ./script.sh

   If the time reported is < than 2 times the time you need to run your app on the whole system, you might have some NUMA issue.
4. A way to avoid the NUMA effect is to use MPI on the top of your OpenMP. With MPI you can bind each MPI process to a dedicated NUMA node. Then the memory will be allocated locally.
5. Another way to solve the NUMA problems is to implement the "First touch policy". This technic consists in doing a "fake" initialization of the data after allocation. This initialization must be done in parallel and must use the same OMP splitting as the one you are using for the computation. It also only works if you are using a static scheduling:

```
int* array = (int*) malloc(SIZE);
#pragma omp parallel for
for(int i=0;i<N;i++)
  array[i]=0;

for(int i=0;i<N;i++)
  //work on array[i]
```

# Activity 1.6 – (Bonus) Managing Flops and tripcount overhead (20mn)

Based on your application, Advisor can take a long time to analyze the flops and trip count. This process uses instrumentation and has a significant overhead. It is possible to use the ITT API to control which part of the code will be collected during the tripcount and flop analysis.

1. Create a new version using the script createnewversion.pl
   $ **iso3dfdlab/createnewversion.pl**
2. Here we only want to profile what happens in the wave propagation iterations. We don't want to collect data that would come from initialization for example during the flop and tripcount analysis.
   Open iso3dfd_main.cc and modify the file by adding the __itt_resume and __itt_pause calls as follow:

```
wstart = walltime();
__itt_resume();
 iso_3dfd(p.next, p.prev, p.vel, coeff, p.n1, p.n2, p.n3, p.num_threads,
p.nreps, p.n1_Tblock, p.n2_Tblock, p.n3_Tblock);
__itt_pause();
 wstop =  walltime();
```

3. You also need to include **<ittnotify.h>**
4. Open the compilation script and add the library *ibittnotify.a* to the linking process. You also need to specify few paths in the compilation script:
   a. Add *<install_dir>/include* to your INCLUDE path for C/C++ applications or *<install_dir>/include/intel64* or *ia32* to your INCLUDE path for Fortran applications
   b. Add *<install_dir>/lib32* to your 32-bit LIBRARIES path
   c. Add *<install_dir>/lib64* to your 64-bit LIBRARIES path
5. Then you can run a tripcount and flop analysis or directly a roofline model to see the difference in the time required for doing the collection.