oneAPI DevSummit hands-on

contents

- Intel Devcloud 접속 환경설정 점검
- CUDA 코드를 SYCL로 변환하기
- Vtune 사용하여 profiling 및 oneMKL를 활용한 최적화

- WSL 설치
  https://learn.microsoft.com/en-us/windows/wsl/install

- Intel DevCloud 접속 가이드 (PVC 노드 접속/사용)
  https://github.com/bjodom/idc

- Training Portal for Intel® Developer Cloud
  https://developer.intel.com/GetStartedDevCloud

- IDC 서버 접속

터미널에서

ssh u******@idcbetabatch.eglb.intel.com

Ex) 강사의 ID : u106300

* Summary slurm command

- slurm is scheduler on IDC
- sinfo : information about system status

  ex) sinfo –al
- srun : submit job

  ex) srun –p <*partition name*> -w <*node name*> --pty bash

* Software Prerequisites

Certain CUDA header files may need to be accessible to SYCLomatic [Supported CUDA version 11.8.]

A. Install on WSM and ‘CUDA SDK’s include directory’ to ‘your home directory of IDC’.

$ wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda_11.8.0_520.61.05_linux.run

$ sh cuda_11.8.0_520.61.05_linux.run

$ tar cvf cuda_inc.tar *<cuda-include-dir>*

$ sftp idcbeta

> put cuda_inc.tar

$ ssh idcbeta

$ tar xvf cuda_inc.tar

B. install ‘CUDA SDK’s include directory’

$ wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda_11.8.0_520.61.05_linux.run

$ sh cuda_11.8.0_520.61.05_linux.run

1. SYCLomatic Practice

- Download mini-nbody

u*****@ idc-beta-batch-head-node:~$ git clone https://github.com/harrism/mini-nbody

- Mode to pvc-node (example)

u*****@ idc-beta-batch-**head-node**:~$ srun -p pvc-shared (-w idc-beta-batch-pvc-node-01) --pty bash

- setup oneAPI

u*****@ idc-beta-batch-**pvc-node-01**:~$ source /opt/intel/oneapi/setvars.sh

```
:: oneAPI environment initialized ::
```

- CPU 에서 컴파일 및 수행

u*****@idc-beta-batch-pvc-node-01:~$ icx -O3 -fopenmp -DSHMOO –lm -o nbody nbody.c

u*****@ idc-beta-batch-pvc-node-01:~$ ./nbody 65536

65536, 46.703

(숫자가 클수록 성능이 좋음)

SYCLomatic mini-nbody on PVC

~$ cd mini-nbody/cuda/

~$ dpct --cuda-include-path=*<CUDA INCLUDE DIR>*
    --extra-arg="-I/usr/include/c++/11"
    --extra-arg="-I/usr/include/x86_64-linux-gnu/c++/11/"
    --extra-arg="-I../" --extra-arg="-DSHMOO" nbody-orig.cu

Compile and Run mini-nbody on PVC

~$ cd dpct_output

~$ icpx -fsycl --verbose nbody-orig.dp.cpp -o nbody-orig-sycl-pvc-exe -I../../ -DSHMOO

~$ ./nbody-orig-sycl-pvc-exe 65536

65536, 239.385

(CPU 보다 ???배 큰 숫자)

2. Vtune 및 mkl을 활용한 최적화

튜토리얼 자료

https://cdrdv2-public.intel.com/671192/mkl-2017-tutorial-fortran.pdf

~$ cp /home/u106300/mkl-fortran-samples-102018.tgz .

~$ tar xvf mkl-fortran-samples-102018.tgz

< 코드 확인 >

두개의 Matrix 곱하기 프로그램

오리지날 코드 확인

~$ cat src/matrix_multiplication.f

< compile >

~$ cd /mkl_fortran_samples/matrix_multiplication

<Makefile 편집>

FC := ifort -> FC := ifx

LIBFLAGS := -mkl -static-intel -> LIBFLAGS := -qmkl

~$ make

< 프로그램 실행 및 Vtune 실행 >

~$ release/matrix_multiplication

.....

 == Matrix multiplication using triple nested loop ==

 == completed at   143.83056 milliseconds ==

 Example completed.

~$ vtune -collect hotspot release/matrix_multiplication

* -collect [hpc-performance | memory-access | hotspot ….]

vtune 수행 (Hotspots)

- Vtune에서 소스코드를 연결해서 보려면 '-g' 옵션을 넣고 컴파일을 해야 한다.

```
tornado@tornado-linux:~/WORK/KSC23$
tornado@tornado-linux:~/WORK/KSC23$ ifx -O3 -g -qmkl -o matrix_multiplication  matrix_multiplication.f
tornado@tornado-linux:~/WORK/KSC23$
```

- vtune –collect *hotspots matrix_multiplication*

```
tornado@tornado-linux:~/WORK/KSC23$
tornado@tornado-linux:~/WORK/KSC23$ vtune -collect hotspot ./matrix_multiplication
vtune: Collection started. To stop the collection, either press CTRL-C or enter from another co
nsole window: vtune -r /home/tornado/WORK/KSC23/r000hs -command stop.
 This example measures performance of computing the real
 matrix product C=alpha*A*B+beta*C using
 a triple nested loop, where A, B, and C are matrices
 and alpha and beta are double precision scalars
```

- 실행 후 아래처럼 100% 라고 나오면, vtune 작업이 끝나며, (hotspot 경우) r001hs 라는 디렉토리가 생성됨

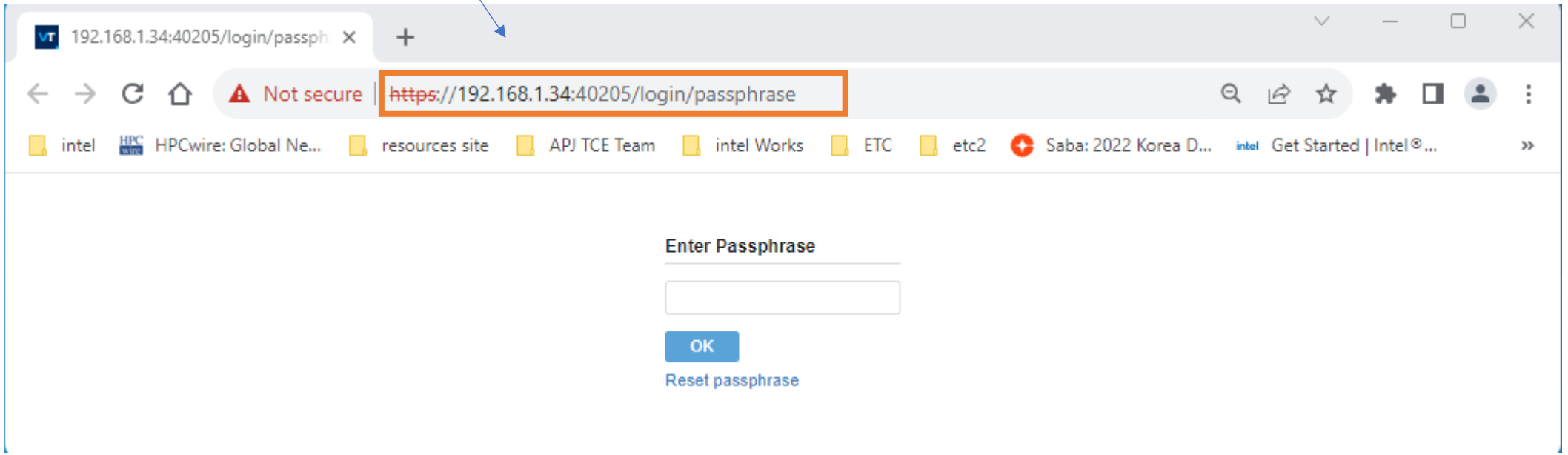
```
vtune: Executing actions 100 % done
tornado@tornado-linux:~/WORK/KSC23$ ls
compile.txt  matrix_multiplication  matrix_multiplication.f  r000hs
tornado@tornado-linux:~/WORK/KSC23$
```

서버에서 vtune-backend 실행

명령어 : vtune-backend -–allow-remote-access –-data-directory (프로파일 결과 디렉토리)

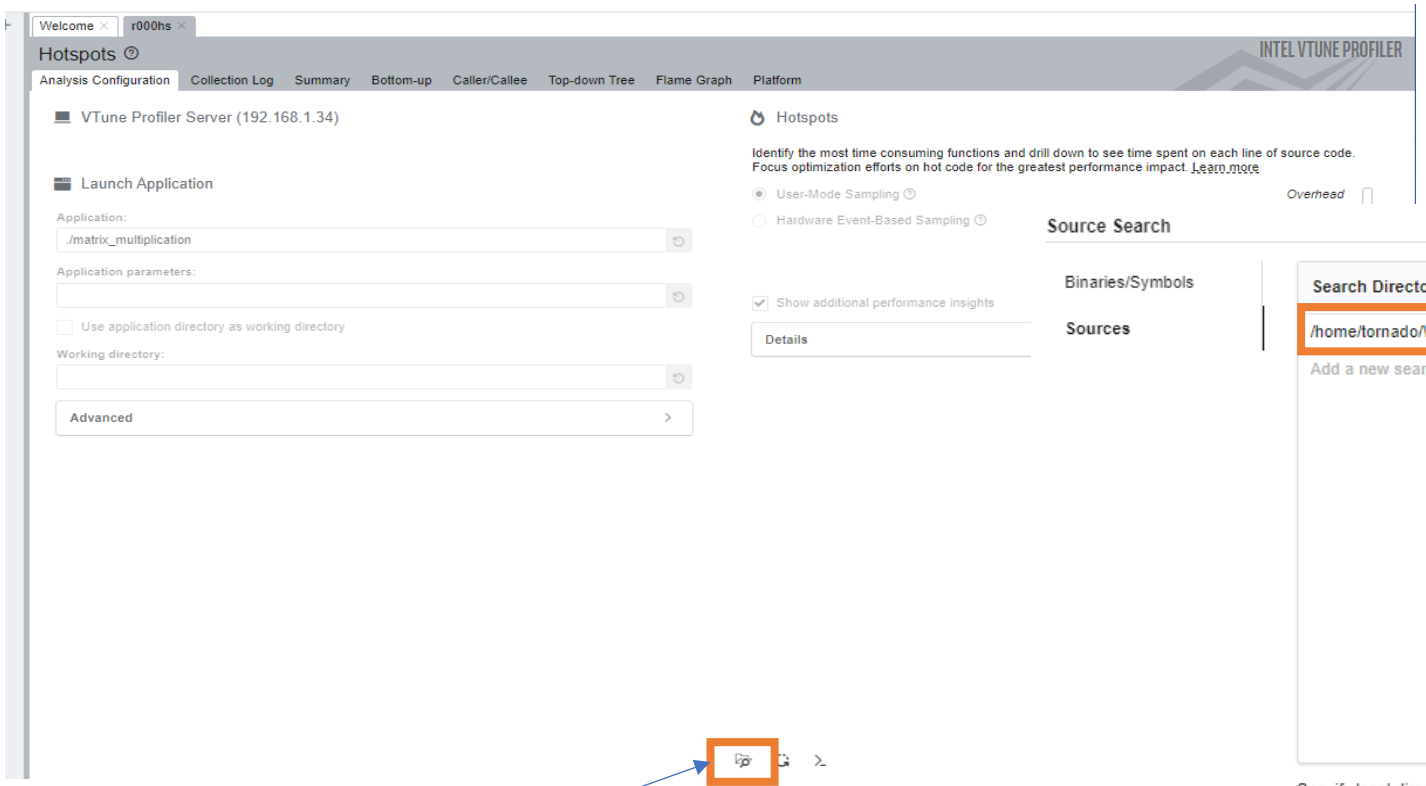
```
tornado@tornado-linux:~/WORK/KSC23$ vtune-backend --allow-remote-access --data-directory /home/
tornado/WORK/KSC23/r00hs
No TLS certificate was provided as a --tls-certificate command-line argument thus a self-signed
 certificate is generated to enable secure HTTPS transport for the web server: /home/tornado/.i
ntel/amplxe/settings/certificates/middleware.crt.
Serving GUI at https://192.168.1.34:40205/
```

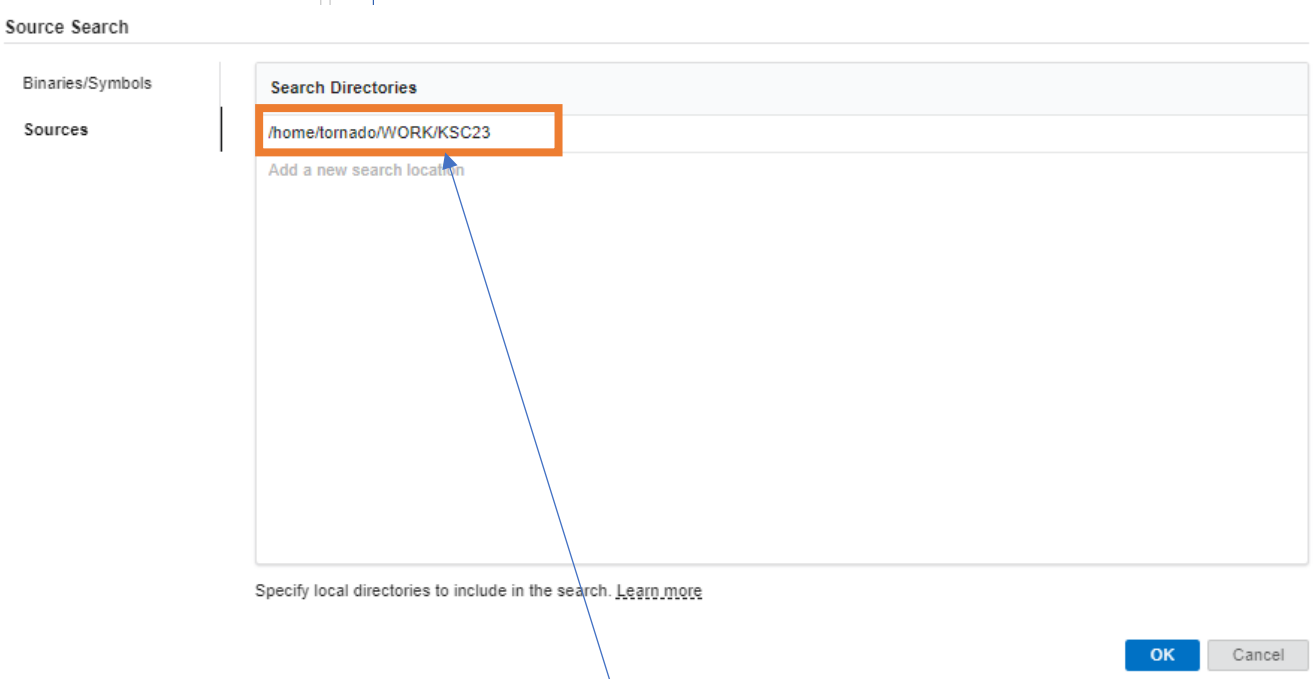
보여지는 URL로 접속



* password 요구시 계정 password를 치고 들러가면 됨

vtune에서 소스코드 연결





Analysis Configuration 탭에서 버튼 클릭

창이 나오면 Sources 탭을 클릭 후, 소스 디렉토리(전체 경로) 입력

결과 1

Welcome | r000hs

Hotspots

INTEL VTUNE PROFILER

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Platform

Elapsed Time: 3.822s

- CPU Time: 3.440s
- Total Thread Count: 1
- Paused Time: 0s

Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee or the Flame Graph view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism : 11.3%

Use Threading to explore more opportunities to increase parallelism in your application.

Microarchitecture Usage : 38.2%

Use Microarchitecture Exploration to explore how efficiently your application runs on the used hardware.

Vectorization : 0.0%

Use HPC Performance Characterization to learn more on vectorization efficiency of your application. This code has floating point operations and is not vectorized. Compiler either recompiling the code with optimization options allow vectorization or using Intel Advisor to vectorize the loops.

INSIGHTS

Top Hotspots

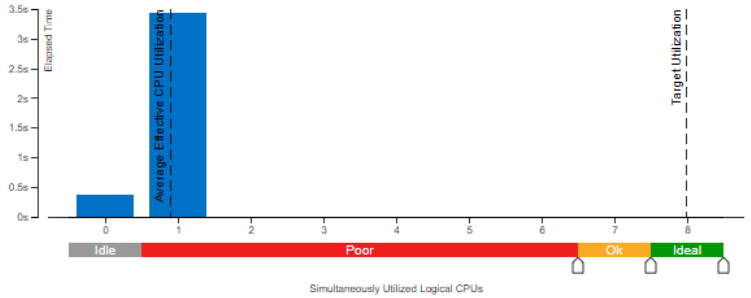
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
MAIN_	matrix_multiplication	3.148s	91.5%
[MKL SERVICE]@dsecnd	libmkl_core.so.2	0.192s	5.6%
func@0x1f1f60	libmkl_core.so.2	0.080s	2.3%
func@0x2042b0	libmkl_core.so.2	0.020s	0.6%

*N/A is applied to non-summable metrics.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

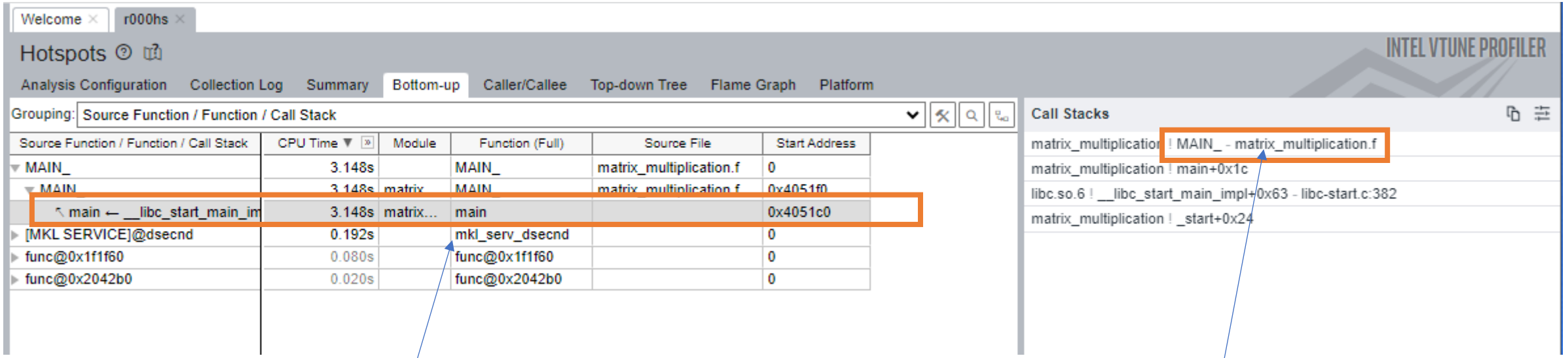


Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: ./matrix_multiplication
Operating System: 6.2.0-26-generic DISTRIB_ID=Ubuntu DISTRIB_RELEASE=22.04 DISTRIB_CODENAME=jammy DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
Computer Name: tornado-linux
Result Size: 3.6 MB
Collection start time: 07:13:54 18/08/2023 UTC
Collection stop time: 07:13:57 18/08/2023 UTC
Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples to speed up post-processing.

결과 2



Bottom-up 탭에서 가장 오래 걸리는 부분 보기

클릭하면 소스코드 탭이 생기며,
가장 시간이 많이 걸리는 라인으로 이동

결과 3

Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Platform | matrix_multiplication.f

Source | Assembly

Source Line	Source	CPU Time: Total	CPU Time: Self
75	DO J = 1, N		
76	TEMP = 0.0		
77	DO K = 1, P		
78	TEMP = TEMP + A(I,K) * B(K,J)	3.6%	0.124s
79	END DO	4.8%	0.166s
80	C(I,J) = TEMP		
81	END DO		
82	END DO		
83			
84	PRINT *, "Measuring performance of matrix product using "		
85	PRINT *, "triple nested loop"		
86	PRINT *, ""		
87	S_INITIAL = DSECND()		
88	DO R = 1, LOOP_COUNT		
89	DO I = 1, M		
90	DO J = 1, N		
91	TEMP = 0.0		
92	DO K = 1, P		
93	TEMP = TEMP + A(I,K) * B(K,J)	17.2%	0.590s
94	END DO	65.6%	2.256s
95	C(I,J) = TEMP	0.3%	0.012s
96	END DO		
97	END DO		
98	END DO		
99	S_ELAPSED = (DSECND() - S_INITIAL) / LOOP_COUNT		
100	PRINT *, "== Matrix multiplication using triple nested loop =="		
101	PRINT 50, " == completed at ",S_ELAPSED*1000," milliseconds =="		
102	50 FORMAT(A,F12.5,A)		
103	PRINT *, ""		
104			
105	IF (S_ELAPSED < 0.9/LOOP_COUNT) THEN		
106	S_ELAPSED=1.D0/LOOP_COUNT/S_ELAPSED		
107	K=(S_ELAPSED*LOOP_COUNT)+1;		

소스코드 탭이 생기며, 94 주변 라인이 65.6% 소요됨

< mkl 코드 확인 >

~$ cat src/dgemm_with_timing.f

```
      PRINT *, "Making the first run of matrix product using "
      PRINT *, "Intel(R) MKL DGEMM subroutine to get stable "
      PRINT *, "run time measurements"
      PRINT *, ""
      CALL DGEMM('N','N',M,N,P,ALPHA,A,M,B,P,BETA,C,M)

      PRINT *, "Measuring performance of matrix product using "
      PRINT *, "Intel(R) MKL DGEMM subroutine"
      PRINT *, ""
      S_INITIAL = DSECND()
      DO R = 1, LOOP_COUNT
          CALL DGEMM('N','N',M,N,P,ALPHA,A,M,B,P,BETA,C,M)
      END DO
      S_ELAPSED = (DSECND() - S_INITIAL) / LOOP_COUNT
      PRINT *, "== Matrix multiplication using Intel(R) MKL DGEMM =="
      PRINT 50, " == completed at ",S_ELAPSED*1000," milliseconds =="
50    FORMAT(A,F12.5,A)
      PRINT *, ""
```

< MKL 적용 코드 성능 확인 >

~$ release/dgemm_with_timing

….

```
 == Matrix multiplication using Intel(R) MKL DGEMM ==
 == completed at      0.31979 milliseconds ==

 It is highly recommended to set parameter LOOP_COUNT
 for this example on your computer as         3128  to have
 total execution time about 1 second for reliability
 of measurements

 Example completed.
```

<참고> oneMKL openMP GPU offload

- openMP GPU offload 설명

https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2023-0/offloading-onemkl-computations-onto-the-gpu.html

Notices and Disclaimers

- You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
- The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that code included in this document is licensed subject to the Zero-Clause BSD open source license (OBSD), http://opensource.org/licenses/0BSD.
- You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
- The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

- Training Portal for Intel® Developer Cloud

https://www.intel.com/content/www/us/en/developer/tools/devcloud/prelaunch-services.html