User Guide

Version 2.5

About CAPC

CAPC is innovative software which provides a fast and effective solution for code parallelizing problems faced by the programmers. It liberates the programmer from the complexity of learning new languages for programming multi-many-cores and GPUs by automatically converting a sequential C code to a corresponding parallel code suitable for the target parallel hardware.

CAPC 2.5 has the following sub products:

1. C to OpenMP3.0 – Automatic Parallelizer for multicore CPUs
2. C to OpenCL – Automatic Parallelizer for OpenCL compatible compute device
3. C to OpenMP 4.5 - Automatic Parallelizer for GPUs
4. C to OpenACC – Automatic Parallelizer for OpenACC compatible devices

Features in CAPC 2.5

- Automatically parallelizes the sequential code to the target parallel language without any inputs/hints from the user
- Support for multiple target Parallel Languages – OpenMP 3.0/OpenCL/OpenMP 4.5/OpenACC
- Single interface for conversion to any parallel language
- Output parallel code is in human readable format that can be further analyzed and optimized

Setting the environment

- Copy the folder CAPC2.5 (from /home/apps/CAPC) to your home directory

  $cd /home/ext/apps/capc

  $cp -r CAPC 2.5 $HOME ($HOME is the location of your home directory)
- Change the CAPC_HOME location in env.sh to point to the location of CAPC2.5 ($CAPC_HOME is the location where you have copied CAPC2.5)

  $CAPC_HOME=$HOME/CAPC2.5
- Then source the $CAPC_HOME/env.sh file to make it available in the terminal

  $source env.sh
- For automatic parallelization of sequential codes to OpenACC/OpenMP 4.5, source the env_gpu.sh file

  $source env_gpu.sh

Usage

The command ***capc*** can be used along with the appropriate flag for parallelizing the sequential C code to the desired parallel language as below:

$capc -c2cl <input_C_program>	For parallelizing C code to OpenCL code for execution on OpenCL compatible devices
$***capc -c2omp <input_C_program>***	For parallelizing C/C++ code to OpenMP 3.0 code for execution on multicore CPUs
$***capc -gpu-omp <input_C_program>***	For parallelizing C/C++ code to OpenMP 4.5 code for execution on GPUs
$***capc -gpu-omp <input_C_program>***	For parallelizing C/C++ code to OpenACC code for execution on OpenACC compatible devices

The output files will be generated inside the $CAPC_HOME/outputs folder.

Testing CAPC

- Find sample C/C++ source codes inside the examples directory
- For parallelization to OpenMP 3.0
    - Invoke the ***capc*** command for converting the C source code to required parallel source code

        $cd $CAPC_HOME/examples/c2omp

        $capc –c2omp 3mm.c
    - The output files (OpenMP file) will be generated inside the $CAPC/outputs folder.
    - You can compile the generated OpenMP code (3mm_c2omp.c) with the following command

        $gcc -fopenmp 3mm_c2omp.c
    - To execute the compiled OpenMP3.0 code, you can submit it as a SLURM job script with the following command. Sample SLURM job script **run_cpu.sh** is kept inside the outputs folder for reference.

        $sbatch run_cpu.sh

- For parallelization to OpenMP 4.5
    - Invoke the ***capc*** command for converting the C/C++ source code to required parallel

source code

$cd $CAPC_HOME/examples/c2omp4.5

$capc -gpu-omp 3mm.c

- The output files (OpenMP 4.5 file) will be generated inside the $CAPC/outputs folder.
- To compile the generated OpenMP4.5 code (eg. 3mm_gpu-omp.c), you can use the clang compiler or the gcc compiler.

  $clang -fopenmp -fopenmp-targets=nvptx64 3mm_gpu-omp.c
- To execute the compiled OpenMP 4.5 code, you can submit it as a SLURM job script requiring GPU nodes. Sample SLURM job script slurm_gpu.sh is kept inside the outputs folder for reference.

  $sbatch slurm_gpu.sh

- For parallelization to OpenACC
  - Invoke the ***capc*** command for converting the C/C++ source code to required parallel source code

    $cd $CAPC_HOME/examples/c2acc

    $capc -gpu-acc 3mm.c
  - The output files (OpenACC file) will be generated inside the $CAPC/outputs folder.
  - To compile the OpenACC code (e.g. 3mm_acc.c), you can use the pgcc compiler

    $pgcc -ta=tesla:cc70 -Minfo=all 3mm_gpu-acc.c
  - To execute the compiled OpenACC code, you can submit it as a SLURM job script requiring GPU nodes. Sample SLURM job script slurm_gpu.sh is kept inside the outputs folder for reference.

    $sbatch slurm_gpu.sh

Limitations

- Dynamic array declarations are not accepted
- Arrays should always be mentioned with their subscript values
- Passing array as a function argument is not supported
- Functions used inside the “for loops” other than Math functions are not supported.