ppOpen-HPC:
Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT).

ppOpen-AT

ver. 1.0.0

User's guide

License

This software is one of the results of the JST CREST ``ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT)" project.

Change History

2012.Nov. Version 0.1 was released.

2013.Nov. Version 0.2 was released.

2016.Mar. Version 1.0.0 is released.

Changes in release 0.2.0

The following functions were added.

- ✓ Added loop split and fusion functions. See section 4, pp. 31 in this manual.
- ✓ Added a function for re-ordering of sentences. See section 4, pp.33 in this manual.
- ✓ Fixed some bugs for adapting code in ppOpen-APPL/FDM and ppOpen-APPL/BEM version 0.1.0.

1. Software Overview

ppOpen-AT is a domain specific language (set of directives) with features that reduce the workload of developers of libraries with auto-tuning features. Specifically, it is a scripting language (set of directives) whose purpose is to auto-generate code required for auto-tuning by placing annotations in the source program, in order to reduce the workload of library developers.

Although the concept of a library with auto-tuning features is applicable to a wide range of processes, the current specification is limited to a language specification specialized for parallel-calculation processing.

2. Target Languages

ppOpen-AT is a scripting language (set of directives) designed to make developing parallel-computation libraries more efficient. Consequently, it must have an interface from a language that is suited to numerical calculation and parallel processing.

The purpose of *ppOpen-AT* is the generation of executable program code. For all of these reasons, it is assumed that *ppOpen-AT* will be used to generate program code that will run in an environment in which Fortran90 or C is available as a language for numerical calculation, and Message Passing Interface (MPI) and/or OpenMP are available as a language for parallel processing.

3. Sequence of Library Development and Use

Fig. 1 illustrates the sequence of library development using *ppOpen-AT*, and the use of mathematics libraries with auto-tuning features by the user.

Please refer to the theory manual if you would like to know more details of *ppOpen-AT*.

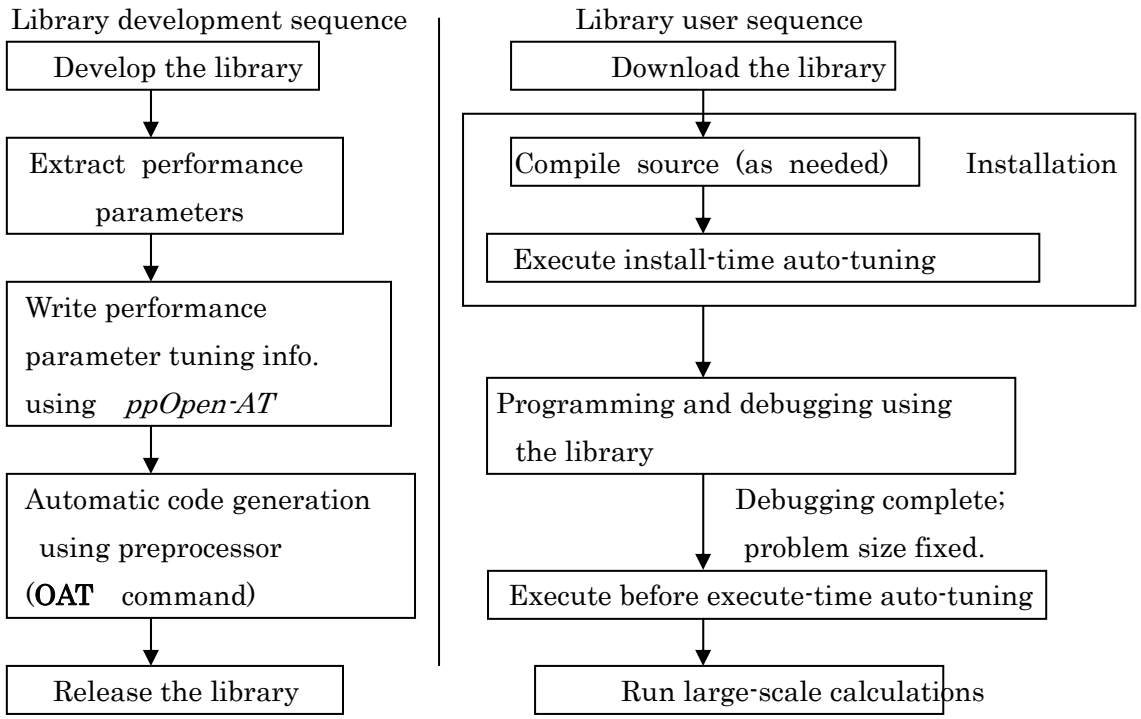

Figure 1. Sequence of the Actions of Library Developers and Users Using *ppOpen-AT*.

4. Directory Organization

The directory organization for the release is summarized as follows.

```
=========================================================================
Directories:
  bin/          Contains executable code
  doc/          Contains the ppohAT user manual
  etc/          Contains miscellaneous files (For Makefile.in.*)
  examples/     Contains sample data for automatic code generation
  include/      Contains header files for external use
  lib/          Contains lib files created for external use (For future extensions)
  src/          Contains the source files of ppohAT
  tools/        Contains the source files of ppohAT tools (For future extensions)
=========================================================================
```

5. System Requirements

To compile the preprocessor of ppOpen-AT, a cpp complier is required. In the current phase, we have confirmed that compilation is successful with the following compilers.

Compilers: gcc, HITACHI, Fujitsu, Intel and PGI.

6. How to Install The Software

The procedure required for installation is as follows.

1. Modify the Makefile.in file

The default Makefile.in is set to operate in a gcc environment. You can utilize one of the following Makefile.in files, if you like:

- ./etc/Makefile.in.hitachi : For the HITACHI C++ compiler.
- ./etc/Makefile.in.fujitsu : For the Fujitsu C++ compiler.
- ./etc/Makefile.in.intel : For the Intel C++ compiler.
- ./etc/Makefile.in.pgi : For the PGI C++ compiler.

Changing the compilers that ppohAT uses to build itself.

The meaning of environment variables are as follows:

- CC - The C++ compiler to be used
- CFLAGS - The compile flags to be passed to the C++ compiler
- COPTFLAGS - The compiler optimization level indicator
- CINCDIR - The header files directory for the C compiler

- LD - Linker
- LDFLAGS - The linker flags for all compilers
- LIBS - The libraries for all compilers
- LIBDIR - The library directories for all compilers

2. Build the archive

Simply enter "make" in the top directory of the archive, then the executable programs will be created under the bin directory. You may also use following make options:

- all - build the entire software package
- src - build the package without examples or tools
- clean - clean out the build tree
- distclean - clean out the build tree and all objects created
- install - copy executables, header files, and libraries to <PREFIX>
- uninstall - remove all installed files and directories

The Installation procedure is simply as follows:

```
$ make
$ make install
```

6. Examples

You can check the status of the build software using a sample program.

In the example folders in the source code, there are two kinds of code that can be processed for ppOpenAT:

- examples/C : For C sample code.
- examples/F90 : For F90 sample code.

The procedure is as follows:

1. Enter the (?) folders that you want to check.
2. Invoke the preprocessor and specify the code file.

For example, if you chose F90 programs, and the unroll function for the matrix-matrix multiplication program, then the following is the procedure.

```
$ cd ./examples/F90/Test_ijk_1_to_4
$ ../../../bin/oat matmul1.f
```

If the installation finishes successfully, you will see the following output:

```
--- ppOpenAT version 1.0.0 ---
OAT_PATH = ""
CommandLine = ../../../bin/oat matmul1.f
```

```
**** Start Code Generation ****
SrcFile ="matmul1.f"
Pass1 Start
Pass2 Start
------ Val List ---------
0: (MIdx=0) Int iauto (RW=1,1) DefPos=13
1: (MIdx=0) Int N=500.000000 (RW=6,0) DefPos=16
2: (MIdx=0) Real*8 A(500)(500) (RW=1,0) DefPos=28
3: (MIdx=0) Real*8 B(500)(500) (RW=1,0) DefPos=35
4: (MIdx=0) Real*8 C(500)(500) (RW=1,0) DefPos=42
5: (MIdx=0) Int ierr (RW=4,0) DefPos=-11
6: (MIdx=0) Int MPI_COMM_WORLD (RW=2,0) DefPos=-11
7: (MIdx=0) Int myid (RW=1,0) DefPos=-11
8: (MIdx=0) Int nprocs (RW=1,0) DefPos=-11
9: (MIdx=0) Int in (RW=1,1) DefPos=-11
10: (MIdx=1) Int MatMul (RW=0,0)(Arg) DefPos=-1
11: (MIdx=1) Real*8 A(0)(0) (RW=1,2)(Arg) DefPos=233
12: (MIdx=1) Real*8 B(0)(0) (RW=1,1)(Arg) DefPos=240
13: (MIdx=1) Real*8 C(0)(0) (RW=1,1)(Arg) DefPos=247
14: (MIdx=1) Int N (RW=13,0)(Arg) DefPos=228
15: (MIdx=1) Real*8 da1 (RW=2,2) DefPos=262
16: (MIdx=1) Real*8 da2 (RW=0,0) DefPos=264
17: (MIdx=1) Real*8 dc (RW=1,1) DefPos=269
18: (MIdx=1) Int i (RW=8,3) DefPos=-11
19: (MIdx=1) Int j (RW=8,3) DefPos=-11
20: (MIdx=1) RealFunc dble (RW=2,0) DefPos=-11
21: (MIdx=1) Int k (RW=2,1) DefPos=-11
Pass3 Start
Pass4 Start
OAT variable:       OAT_DEBUG = 1
OAT variable:       OAT_NUMPROCS = 4
OAT variable:       OAT_STARTTUNESIZE = 100
OAT variable:       OAT_ENDTUNESIZE = 500
OAT variable:       OAT_SAMPDIST = 100
Base Parameter (BPset): N
------ Tuning Region -- Count = 1
```

```
Tuning Region : 0
Install Unroll MyMatMul
OAT_InstallMyMatMul(,,N,A,C,B)
Undefined Val ( use module ) = (i j k )
    varied i (1,N,) from 1.000000 to 4.000000 step 1.000000
    varied j (1,N,) from 1.000000 to 4.000000 step 1.000000
    varied k (1,N,) from 1.000000 to 4.000000 step 1.000000
  RefValStr = ( i,j,da1,k,dc, )
  Case Count = 64

Pass5 Start
TuneRegion = 1/1
**** End Code Generation **** OK !

After that, the folder "OAT" is generated.
The "OAT/*.f" code is generated by ppOpen-AT automatically.
```