Compilertools Documentation

Release 1.1.2

J.Goutin

Feb 14, 2021

User Documentation:

Python uses the Wheel format for simplified package distribution. However, it does not allow to distribute packages optimized for each machine but highly compatible ones. The user must compile the package himself to take advantage of optimization like SIMD (SSE, AVX, FMA, ...).

Compilertools allows to work around this problem and distribute optimized packages for several machines while keeping the simplicity of Wheel. It works in the background and has been created with the aim of being easy to use. Package maintainer requires only to import it at runtime and buildtime. Everything is transparent for the end user.

Its secondary objective is also to help the package maintainer to optimally compile its package with multiple compilers by configuring options for him.

CHAPTER 1

Features:

- Compiles and distributes optimized binaries for a variety of machines in a single Wheel package.
- Helps to build optimized package from sources for current machine.
- Handles automatically compiling and linking options for a variety of compilers.
- Autodetects openMP, OpenACC or Intel Cilk Plus in source code and automatically sets related compiling and linking options.
- Support extra compiling options like fast math.
- Provides build time settings for package maintainer to tweak compilation.
- Provides API for getting information on current machine CPU.
- Lightweight pure Python module with no dependency that use lazy import and evaluation as possible.

CHAPTER 2

How that works ?

Compilertools dynamically sets link options and compile options depending on the currently used compiler and targeted architecture.

This avoid to have to specify compiler specific options in sources or setup files.

2.1 Multi-architecture optimized compilation for distribution

At build time:

Compilertools helps to make optimized ".so"/".pyd" for each architecture and name files with tagged suffixes :

Example:

- *module.avx2.cp36-win_amd64.pyd* -> Optimized variant for AVX2 SIMD Extensions
- *module.avx.cp36-win_amd64.pyd* -> Optimized variant for AVX SIMD Extensions
- *module.cp36-win_amd64.pyd* -> Classic highly-compatible variant

These optimized files are packaged in the same wheel when distributing (Don't need to create a wheel by variant).

Requirement:

- Import `compilertools.build` before build module normally.
- Options can easily be changed directly in `compilertools.build.CONFIG_BUILD` dictionary.

At runtime:

Compilertools detects and chooses the best optimized ".so"/".pyd" to run based on CPU information.

If the best file not exist, search for the second best file, etc... If nothing found, use the highly-compatible one.

Requirement:

- Import `compilertools` one time before import optimized modules (This add a new import hook to Python).

2.2 Current-architecture optimized compilation

Compilertools finds the best compiler options for the current architecture and current compiler and build only one optimized ".so"/".pyd" with classic name.

Requirement:

- Import `compilertools.build` before build module normally.

2.3 And also...

openMP, OpenACC, Intel Cilk Plus API auto-detection: Compilertools searches in source files for API `pragma` preprocessor calls and enables compiler and linker options if needed.

Extra generic compilers options: Compilertools can enable or disable generic extra compiler options like fast math.

CHAPTER 3

Compatibility

3.1 Supported Compilers

Compilertools implements support for following compilers:

- GCC
- LLVM Clang
- Microsoft Visual C++

3.2 Supported Processors

Compilertools implements support for following CPU:

- x86 (32 and 64 bits)

3.3 Build tools compatibility

Compilertools have been tested with following build tools:

- Distutils
- Setuptools
- Numpy.distutils
- Cython

CHAPTER 4

Documentation

4.1 Getting Started

4.1.1 Installation

Compilertools is available on PyPI and can be installed with pip:

```
pip install compilertools
```

4.1.2 Enabling compilertools in a Python package

This explains how to enable compilertools in a Python package:

Enabling compilertools import hook

The import hook is used to select the best version of a compiled module on build time. It needs to be initialized on module startup.

This can be done by simply importing `compilertools` on the top of the main module of the package (`__init__.py` depending of package architecture):

```
"""Main package __init__.py"""
try:
    import compilertools
except ImportError:
    # Reverts back to classic import behavior if compilertools not available
    # or imported on a non compatible Python version.
    pass
```

The `try except` block ensures that the package is imported correctly even if compilertools is not available or can't run. In this case, compiled modules will be imported in compatibility mode, without optimisations.

Enabling compilertools on build

To generate multiple optimized compiled modules, compilertools needs to be initialized in the `setup.py` of the package.

This can be done by simply importing `compilertools.build` in setup.py just after used build library (like `setuptools`, `distutils`, ...):

```
"""setup.py file"""
from setuptools import setup
try:
    import compilertools.build
except ImportError:
    # Reverts back to classic build behavior if compilertools imported
    # on a non compatible Python version.
    pass
```

Don't forget to add compilertools as requirement for the package in the `install_requires` argument of inside `setup.py`.

The `try except` block ensures that `setup.py` can still be used on Python versions that are not supported by compilertools (Like Python 2). In this case, compiled modules will be build without optimization.

```
"""setup.py file, continued..."""
setup(
    # ...Others setup arguments...
    install_requires=['compilertools']
    )
```

And next ?

That's its, compilertools is enabled on the package. Its configuration can be tweaked (See below) if needed, but it work with default value else.

You can now build the wheel package with `setup.py bdist_wheel`. If an user uses this generated wheel, Python will use the best optimized compiled file available inside the package for its machine.

If an user build your package with `pip` from source, it will get an automatically optimized file for its machine.

4.1.3 Configuring compilertools

Compilertools configuration is done with the `compilertools.build.ConfigBuild` object, simply by changing its parameters to adjust compilertools behavior as needed directly in `setup.py`:

```
"""setup.py file"""
try:
    import compilertools.build

    # Creates optimized modules only for AVX2 and AVX512 CPU instructions.
    compilertools.build.ConfigBuild.suffixes_includes = ['avx2', 'avx512']
except ImportError:
    pass
```

Read *ConfigBuild documentation* for available parameters.

4.1.4 compilertools exception

On import or on installation from PIP, compilertools exceptions are ignored (only logged in stdout) to not break application.

In this case, no optimizations are enabled and module are compiled/loaded in compatible mode.

4.2 Using it as system information library

Compilertools gets system information to optimize build and import. So, it can also be used as library that provides this information.

4.2.1 CPU information

Compilertools can provide information on current CPU (Using `CPUID` or equivalent):

```
import compilertools

# Gets current processor (arch=None for autodetect current CPU architecture)
cpu = compilertools.get_processor()

# Gets features flags for current CPU as property
cpu.features
>>> {'SSE3', 'LAHF_LM', 'PSE36', 'ADX', 'MCA', 'XTPR', 'POPCNT', 'CLFLUSH',
     'DE', 'TSC', 'MSR', 'MTRR', 'SSE4_1', 'F16C', 'TSC_ADJUST', 'INVPCID',
     'ABM', 'SMX', 'SDBG', 'VME', 'FXSR', 'CX16', 'MOVBE', 'DTES64', 'AVX',
     'AVX2', 'ERMS', 'RDTSCP', 'PCID', 'CLFLUSHOPT', 'MCE', 'RDRAND',
     'SMEP', 'TM2', 'SMAP', 'LM', '3DNOWPREFETCH', 'XSAVE', 'DS', 'RTM',
     'PSE', 'TSC_DEADLINE_TIMER', 'FSGSBASE', 'SSE4_2', 'TM', 'PDPE1GB',
     'PAT', 'DS_CPL', 'SSE2', 'FMA', 'VMX', 'BMI2', 'CX8', 'X2APIC', 'HLE',
     'AES', 'EST', 'PAE', 'SSSE3', 'SYSCALL', 'HT', 'MMX', 'SEP', 'PDCM',
     'CMOV', 'SS', 'MONITOR', 'BMI1', 'MPX', 'PCLMULQDQ', 'OSXSAVE', 'NX',
     'SSE', 'APIC', 'PGE', 'FPU', 'ACPI', 'RDSEED', 'PBE'}
```

see *API documentation* for available properties.

4.2.2 Compiler information

Compilertools can provide information on compiler:

```
import compilertools

# Gets current compiler
compiler = compilertools.get_compiler()

# Gets current compiler version as property
compiler.version
>>> 6.3
```

see *API documentation* for available properties.

4.3 API documentation

4.3.1 compilertools.build

Building functions

class compilertools.build.**ConfigBuild**
Build configuration

api = {'cilkplus': {'c': '#pragma simd ', 'fortran': '!dir$ simd '}, 'openacc': {'
Specific API are auto-enabled when compiling and linking if following preprocessors are detected in source files

current_machine = 'autodetect'
Compiles optimized for current machine only (If not compile for a cluster of possibles machines) True or False for manually set value; 'autodetect' for automatically set value to True if build from PIP

disabled = False
Disable compilertools's optimization while building

extensions = {'c': ('.c', '.cpp', '.cxx', '.cc', '.c++', '.cp'), 'fortran': ('.f', '
Sources files extensions for code analysis

option = {'fast_fpmath': False}
Enables compilers options

suffixes_excludes = {'amd', 'intel', 'intel_atom', 'sse', 'sse4_1', 'sse4_2', 'ssse3'}
Disabled suffixes in files matrix definition. If 'suffixes_includes' is empty, completes this set to not build files for a specific architecture. This does not affect current machine builds.

suffixes_includes = {}
Enabled suffixes in files matrix definition. If this set is not empty, includes only suffixes specified inside it. This does not affect current machine builds.

compilertools.build.**get_build_compile_args**(*compiler=None*, *arch=None*, *current_machine=None*, *ext_suffix=None*, *use_option=None*, *use_api=None*)
Gets compiler args for build as a dict of file suffixes as key and args string as values.

Parameters

- **compiler** (*str or compilertools.compilers.CompilerBase subclass*) – compiler name or instance. If None, use distutils default value
- **arch** (*str*) – target architecture name.
- **current_machine** (*bool*) – return only one suffix/args pair optimized for current machine only. If None, use CONFIG_BUILD value.
- **ext_suffix** (*list of str*) – Extensions to use after suffix.
- **use_option** (*list of str*) – List of options to use (fast_fpmath, ...).
- **use_api** (*list of str*) – List of API to use (openmp, ...). If None, don't enable API.

Returns Suffixes are keys, arguments are values.

Return type dict with str as keys and values

compilertools.build.**get_build_link_args**(*compiler=None*, *use_api=None*, *use_option=None*)
Gets linker arg for build as a list of args string.

Parameters

- **compiler** (*str or compilertools.compilers.CompilerBase subclass*) – compiler name or instance. If None, use distutils default value
- **use_api** (*list of str*) – List of API to use (openmp, ...). If None, don't enable API.
- **use_option** (*list of str*) – List of options to use (fast_fpmath, ...).

compilertools.build.**get_compile_args**(*compiler=None*, *arch=None*, *current_machine=False*, *current_compiler=False*)

Gets compiler args OrderedDict for a specific compiler and architecture combination.

Parameters

- **compiler** (*str or compilertools.compilers.CompilerBase subclass*) – Compiler name or instance.
- **arch** (*str*) – Target architecture name.
- **current_machine** (*bool*) – Only compatibles with current machine CPU
- **current_compiler** (*bool*) – If True, return only arguments compatibles with current compiler.

Returns Arguments

Return type collections.OrderedDict

compilertools.build.**get_compiler**(*compiler=None*, *current_compiler=False*)

Returns compiler class

Parameters

- **compiler** (*str of CompilerBase subclass*) – Compiler Name or instance
- **current_compiler** (*bool*) – Compiler used to build

Returns Compiler class instance.

Return type CompilerBase subclass instance

compilertools.build.**suffix_from_args**(*args*, *extension=''*, *return_empty_suffixes=False*)

Returns suffixes from args.

Parameters

- **args** (*collections.OrderedDict*) – Arguments.
- **extension** (*str or list of str*) – File extensions.
- **return_empty_suffixes** (*bool*) – If True, return '' suffixes.

Returns Suffixes

Return type list of str

4.3.2 compilertools.imports

Import machinery

compilertools.imports.**ARCH_SUFFIXES = ['.avx512-intel.cpython-37m-x86_64-linux-gnu.so', '.a**

Current arch compatibles suffixes

compilertools.imports.**update_extensions_suffixes**(*compiler*)

Updates file extensions suffixes compatibles with current machine with ones from a specified compiler.

Parameters **compiler** (*str or None*) – compiler name. If None, uses default compiler name on current platform

4.3.3 compilertools.compilers

Compilers

class compilertools.compilers.**CompilerBase**(*current_compiler=False*)

Base class for compiler

Arg

alias of Argument

api

Compatibles API

Returns Keys are API names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

clear() → None. Remove all items from D.

compile_args(*arch=None*, *current_machine=False*)

Gets compiler args list for a specific architecture.

Parameters

- **arch** (*str*) – Target architecture name.
- **current_machine** (*bool*) – If True, returns only arguments compatibles with current machine (conditions from "Arg.import_if").

Returns Arguments matrix. Keys are suffixes, values are compiler arguments.

Return type collections.OrderedDict with keys and values as str

compile_args_current_machine()

Return compiler arguments optimized by compiler for current machine

Returns Best compiler arguments for current machine.

Return type str

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

name

Compiler type name

Returns Name.

Return type str

option

Compatibles Options

Returns Keys are options names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update(*[E]*, ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

compilertools.compilers.**get_compiler**(*compiler=None*, *current_compiler=False*)
Returns compiler class

Parameters

- **compiler** (*str of CompilerBase subclass*) – Compiler Name or instance
- **current_compiler** (*bool*) – Compiler used to build

Returns Compiler class instance.

Return type CompilerBase subclass instance

compilertools.compilers.gcc

GNU Compiler Collection

class compilertools.compilers.gcc.**Compiler**(*current_compiler=False*)
GNU Compiler Collection

Arg
alias of compilertools.compilers._core.Argument

api
Compatibles API

Returns Keys are API names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

clear() → None. Remove all items from D.

compile_args(*arch=None*, *current_machine=False*)
Gets compiler args list for a specific architecture.

Parameters

- **arch** (*str*) – Target architecture name.
- **current_machine** (*bool*) – If True, returns only arguments compatibles with current machine (conditions from "Arg.import_if").

Returns Arguments matrix. Keys are suffixes, values are compiler arguments.

Return type collections.OrderedDict with keys and values as str

compile_args_current_machine()
Return compiler arguments optimized by compiler for current machine

Returns Best compiler arguments for current machine.

Return type str

get(*k[, d]*) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

name
Compiler type name

Returns Name.

Return type str

option
Compatibles Options

Returns Keys are options names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

python_build_version
Compiler version that was used to build Python.

Returns Version.

Return type float

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

version
Compiler version used.

Returns Version.

Return type float

compilertools.compilers.llvm

LLVM Clang

class compilertools.compilers.llvm.**Compiler**(*current_compiler=False*)
LLVM Clang

Arg
alias of compilertools.compilers._core.Argument

api
Compatibles API

Returns Keys are API names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

clear() → None. Remove all items from D.

compile_args(*arch=None*, *current_machine=False*)
Gets compiler args list for a specific architecture.

Parameters

- **arch** (*str*) – Target architecture name.
- **current_machine** (*bool*) – If True, returns only arguments compatibles with current machine (conditions from "Arg.import_if").

Returns Arguments matrix. Keys are suffixes, values are compiler arguments.

Return type collections.OrderedDict with keys and values as str

compile_args_current_machine()
Return compiler arguments optimized by compiler for current machine

Returns Best compiler arguments for current machine.

Return type str

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

name
Compiler type name

Returns Name.

Return type str

option
Compatibles Options

Returns Keys are options names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

python_build_version
Compiler version that was used to build Python.

Returns Version.

Return type float

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], **F) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

version
Compiler version used.

Returns Version.

Return type float

compilertools.compilers.msvc

Microsoft Visual C++ Compiler

class compilertools.compilers.msvc.**Compiler**(*current_compiler=False*)

Microsoft Visual C++

Arg

alias of compilertools.compilers._core.Argument

api

Compatibles API

Returns Keys are API names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

clear() → None. Remove all items from D.

compile_args(*arch=None*, *current_machine=False*)

Gets compiler args list for a specific architecture.

Parameters

- **arch** (*str*) – Target architecture name.
- **current_machine** (*bool*) – If True, returns only arguments compatibles with current machine (conditions from "Arg.import_if").

Returns Arguments matrix. Keys are suffixes, values are compiler arguments.

Return type collections.OrderedDict with keys and values as str

compile_args_current_machine()

Return compiler arguments optimized by compiler for current machine

Returns Best compiler arguments for current machine.

Return type str

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

name

Compiler type name

Returns Name.

Return type str

option

Compatibles Options

Returns Keys are options names, values are dict of arguments with keys in {'link', 'compile'}.

Return type dict

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update(*[E]*, ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

version
For Microsoft Visual C++, Compiler version used to build need to be the same that the one used to build Python.

Returns Version.

Return type float

4.3.4 compilertools.processors

Processors

class compilertools.processors.**ProcessorBase**(*current_machine=False*)
Base class for CPU

arch
processor architecture

Returns Architecture name.

Return type str

clear() → None. Remove all items from D.

get(*k[, d]*) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k[, d]*) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k[, d]*) → D.get(k,d), also set D[k]=d if k not in D

update(*[E]*, ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

compilertools.processors.**get_processor**(*arch*, **args*, ***kwargs*)
Return processor class

Parameters

- **arch** (*str or None*) – processor architecture
- **kwargs** (*args,*) – args for class instantiation

Returns Processor class instance.

Return type ProcessorBase subclass instance

compilertools.processors.**get_arch**(*arch=None*)
Checks architecture name and returns fixed name.

Parameters **arch** (*str*) – Architecture name to check.

Returns Fixed architecture name.

Return type str

compilertools.processors.x86_32

X86-32 Processors

class compilertools.processors.x86_32.**Processor**(*current_machine=False*)
x86-32 CPU

arch
processor architecture

Returns Architecture name.

Return type str

brand
CPU's brand from CPUID

Returns Brand.

Return type str

clear() → None. Remove all items from D.

cpuid_highest_extended_function
CPUID highest extended function.

Returns Related EAX value for CPUID.

Return type int

features
CPU's features flags from CPUID

Returns Flags names.

Return type list of str

References

Reference: Linux kernel "arch/x86/include/asm/cpufeatures.h"

Feature naming convention: Use "cpufeatures.h" quoted names in comments in priority, then use name from "cpufeatures.h" constants.

Exceptions in names: PNI called SSE3 (like other SSE feature flags)

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

os_supports_xsave
OS and CPU supports XSAVE instruction.

Returns Supports if True.

Return type bool

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

vendor
CPU's manufacturer ID from CPUID.

Returns Manufacturer ID.

Return type str

class compilertools.processors.x86_32.**Cpuid**(*eax_value=0*, *ecx_value=0*)
Gets Processor CPUID.

Parameters

- **eax_value** (*int*) – EAX register value
- **ecx_value** (*int*) – ECX register value

eax
Get EAX register CPUID result.

Returns Raw EAX register value.

Return type int

ebx
Get EBX register CPUID result.

Returns Raw EAX register value.

Return type int

ecx
Get ECX register CPUID result.

Returns Raw EAX register value.

Return type int

edx
Get EDX register CPUID result.

Returns Raw EAX register value.

Return type int

static registers_to_str(**uints*)
Converts unsigned integers from CPUID register to ASCII string.

Parameters **uints** (*int*) – Unsigned integers to concatenate and convert to string.

Returns Result.

Return type str

compilertools.processors.x86_64

x86-64 Processors

class compilertools.processors.x86_64.**Processor**(*current_machine=False*)
x86-64 CPU

arch
processor architecture

Returns Architecture name.

Return type str

brand
CPU's brand from CPUID

Returns Brand.

Return type str

clear() → None. Remove all items from D.

cpuid_highest_extended_function
CPUID highest extended function.

Returns Related EAX value for CPUID.

Return type int

features
CPU's features flags from CPUID

Returns Flags names.

Return type list of str

References

Reference: Linux kernel "arch/x86/include/asm/cpufeatures.h"

Feature naming convention: Use "cpufeatures.h" quoted names in comments in priority, then use name from "cpufeatures.h" constants.

Exceptions in names: PNI called SSE3 (like other SSE feature flags)

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

os_supports_xsave
OS and CPU supports XSAVE instruction.

Returns Supports if True.

Return type bool

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update(*[E], **F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

vendor
CPU's manufacturer ID from CPUID.

Returns Manufacturer ID.

Return type str

class compilertools.processors.x86_64.**Cpuid**(*eax_value=0*, *ecx_value=0*)
Gets Processor CPUID.

Parameters

- **eax_value** (*int*) – EAX register value
- **ecx_value** (*int*) – ECX register value

eax
Get EAX register CPUID result.

Returns Raw EAX register value.

Return type int

ebx
Get EBX register CPUID result.

Returns Raw EAX register value.

Return type int

ecx
Get ECX register CPUID result.

Returns Raw EAX register value.

Return type int

edx
Get EDX register CPUID result.

Returns Raw EAX register value.

Return type int

static registers_to_str(**uints*)
Converts unsigned integers from CPUID register to ASCII string.

Parameters **uints** (*int*) – Unsigned integers to concatenate and convert to string.

Returns Result.

Return type str

4.4 Changelog

4.4.1 1.1.2 (2020/10/03)

Fixes:

- Fix GCC/LLVM version detection in some cases.

Others:

- Use `-O3` instead of `-O2` on GCC/LLVM.
- Ensure Python 3.9 compatibility.
- Drop Python 3.5 support.

4.4.2 1.1.1 (2020/01/29)

Others:

- Ensure Python 3.8 compatibility.
- Drop Python 3.4 support.

4.4.3 1.1.0 (2019/01/11)

New compilers support:

- LLVM Clang

Others:

- Add more easy access function to get current machine CPU and compiler information.
- Use `-O2` instead of `-O3` on GCC.

4.4.4 1.0.0 (2018/05/27)

First version with following features:

Compilers:

- Microsoft Visual C++ compiler
- GNU Compiler Collection

Processors:

- X86 (32/64 bits)
- x86 CPUID

Others:

- Multi-architecture optimized compilation for distribution
- Current-architecture optimized compilation
- API autodetection: openMP, openACC & Intel Cilk Plus
- Extra compilers options: fast math
- Extended configuration
- Build tools compatibility: Distutils, Setuptools, Numpy.distutils & Cython
- PIP autodetection
- Full Unittest with pytest and Continuous integration on Appveyor and Travis-CI

- Sphinx Documentation with Readthedoc

4.4.5 Ideas & possibles futures features

- Ability to force the use of an API (openMP, ...)
- Add it as an option in *setuptools* (or eventually merge it in *setuptools*).
- Since this module get many information on CPU, give possibility to users to access them easily as a dict.
- Add support for more compilers (Intel, LLVM, ...). The aim is to support all compilers available with *distutils* and *numpy.distutils*.
- Add support for more processors (ARM, ...).

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

D

E

F

G

I

K

N

O

P

R

S

U

V