Mettre en place une chaîne de compilation

Le but est de mettre en place une chaîne de compilation complète, multi-plateformes, prenant en charge l'ensemble des étapes de la production de logiciel :

- partage du code et gestion des versions ;
- analyse statique et sémantique du code ;
- compilation du code avec plusieurs compilateurs différents, pour comparer les performances et avoir un maximum de messages d'avertissement et d'erreur plus compréhensible ;
- l'exécution des tests unitaires et la génération de rapports ;+
- l'analyse des performances ;
- la génération de la documentation ;
- le paquetage et le déploiement des applications ;
- le suivi des problèmes rapportés par les utilisateurs.

Intégration continue : les modifications faites par les développeurs passent par un processus de validation en plusieurs étapes, permettant de vérifier les contraintes de développement avant intégration dans l'application finale. Processus automatisé permet de tester très régulièrement (en permanence) les modifications. L'objectif est de détecter très tôt les problématiques, pour les corriger rapidement. Utilisation de serveurs d'intégration continue : CDash, Jenkins, Hudson, TEamcity, CruiseControl ; outils de build : Meaven, Ant, Gradle, Ivy

La première base : le moteur de production

Le moteur de production est un outil qui permet de gérer les projets (liste des fichiers à utiliser, en fonction des options de compilation). Doit pouvoir être utiliser avec différents éditeurs. Nous allons présenter l'utilisation de CMake dans cet ouvrage, n'hésitez pas à tester d'autres

outils (Makefiles, Apache Ant, GNU Autotools, Scons).

Un point qui ne peut pas être fait par les outils, ce sont les choix de conception du code. Il est important de viser dans un premier temps une bonne conception en laissant de côté les problèmes de performances, le but est surtout de pouvoir modifier facilement le code en fonction des mesures de performances. Il sera acceptable dans la majorité des cas d'avoir un code un peu moins performant, mais bien conçu, qu'un code qui serait un peu plus performant, mais dont la moindre évolution prendrait beaucoup de temps. Dans les cas où l'on souhaiterait quand même écrire un tel code, il est primordial d'isoler ces parties de code spécifiques d'une plateforme ou d'un matériel. Le rôle du moteur de production sera aussi de gérer les différents codes possibles.

CMake utilise des fichiers texte (CMakeLists.txt) utilisant un langage déclaratif simplifié (variables, listes, conditions, boucles, macros) pour générer les fichiers de compilation ou projet pour IDE.

```
set(Ma_Variable 1)
set(Ma_Liste "b.c; c.c")

if (Ma_Variable)
    set(Ma_Liste "a.c;${Ma_Liste}")
else (Ma_Variable)
    set(Ma_List "${Ma_Liste};d.c")
endif (Ma_Variable)

foreach (Element ${Ma_Liste})
    message(STATUS "Element: ${Element}")
endforeach (Element)

macro (doPrint Variable)
    message(STATUS "V = ${Variable}")
endmacro (doPrint)
doPrint("coucou")
```

Exemple de CMakeLists.txt. Structure :

```
./computePi
    CMakeLits.txt
```

```
        PROJECT(compute_pi)
        CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
        INSTALL(FILES ./include/compute_pi.h DESTINATION
include)
        INCLUDE_DIRECTORIES(./include)
        SUBDIRS(./src)
    include/
        compute_pi.h
    src/
        compute_pi.c
        main.c
        CMakeLists.txt
            ADD_LIBRARY(compute_pi SHARED "compute_pi.c")
            INSTALL(TARGETS compute_pi LIBRARY DESTINATION
lib)
            ADD_EXECUTABLE(compute_pi "main.c")
            TARGET_LINK_LIBRARIES(computePi compute_pi)
            INSTALL(TARGETS computePi RUNTIME DESTINATION
bin)
```

Exemple de compilation conditionnelle avec pthread

```
// CMakeLists.txt
INCLUDE(FindThreads)
IF (CMAKE_HAVE_PTHREAD_H)
    SET(CMAKE_REQUIRED_LIBRARIES
"${CMAKE_THREAD_LIBS_INIT}")
    LINK_LIBRARIES(${CMAKE_THREAD_LIBS_INIT})
    SET(HAVE_PTHREAD 1)
ELSE (CMAKE_HAVE_PTHREAD_H)
    ...
ENDIF (CMAKE_HAVE_PTHREAD_H)

// Dans congif.h.in
#cmakedefine HAVE_PTHREAD 1
#cmakedefine HAVE_LIB_//_2 1

// config.h, généré par CMake
#define HAVE_PTHREAD 1
/* #cmakedefine HAVE_LIB_//_2 1

// Dans compute_pi.c
```

```
#ifdef HAVE_CONFIG_H
#  include "config.h"
#endif
#if define (HAVE_PTHREAD)
#  include <pthread.h>
#  define CONTEXTE_PARALLELE pthread_t contexte;
#elif defined (HAVE_LIB_//_2)
#  include <lib_parallel2.h>
#  define CONTEXTE_PARALLELE lib_parallel2_t contexte;
#else
#  define CONTEXTE_PARALLELE
#endif
```

Utilisation de pkg-config ?

La seconde base : génération de rapport

Besoin de générer sous forme facilement exploitable, c'est à dire lisible (graphiquement), triable, filtrable, catégories, etc. Par exemple Qt Creator ou CDash. Open Babel Dashborad

Partage du code et gestion des versions

outils : subversion, Git, mercurial

utilisation en local avec un serveur ou via un site publique (github)

fonctionnalités : gestion des versions, des tags, des branches

Compilateurs et débogueur

Au moins les outils gratuits : gcc, clang, intel, microsoft

Débogage : gdb

Analyse statique, sémantique et dynamique du code

Message d'avertissement des compilateurs. Activer un maximum d'informations (à désactiver uniquement fichier par fichier, en cas de code volontairement spécial). Les messages d'avertissement ne sont pas optionnel.

Outils spécifique : permet d'améliorer le code, ce qui peut avoir un impact sur les performances. gprof, Intel Inspector, cppchecker

A voir : leakTracer, MallocDebug, Visual Leak Detector, Clang static analyzer

Tests unitaires

boost, QTest, CTest

Analyse des performances

gprof, intel, studio (VTune), microsoft ? AMD ? Valgrind (callgrind), Shark/Saturn (Mac OS X only)

Génération de la documentation

outils : doxygen, qdoc, docutils

Au minimum : documentation technique, à usage interne. Mais possiblité de générer des documentations complètes (cf Qt), en HTML, PDF, LaTex

Paquetage et déployement

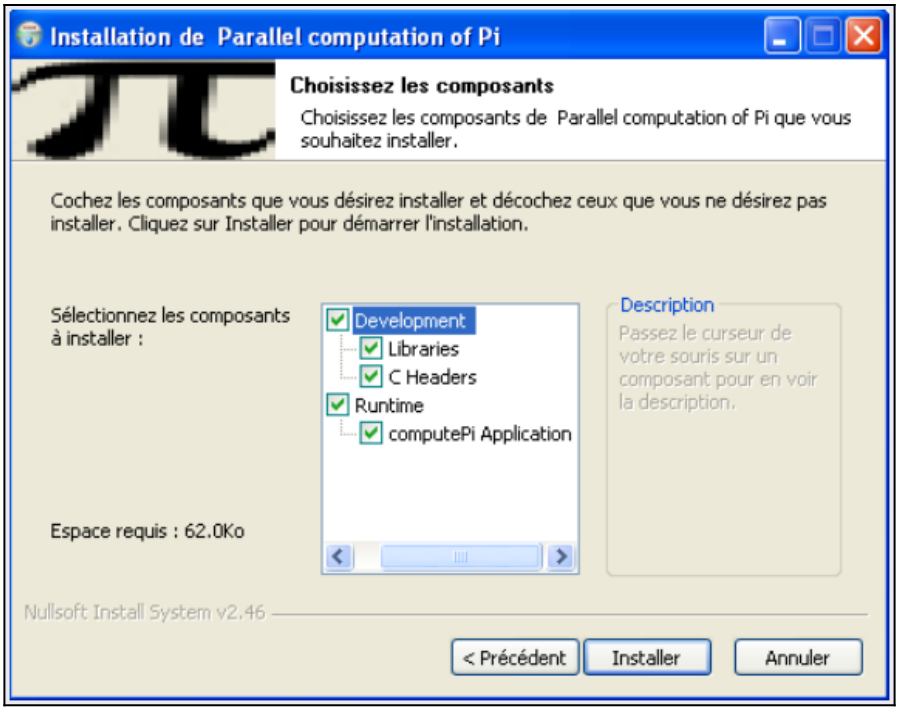
CPack, génération de paquets plateforme spécifique (.deb, .rpm, dépôts Ubuntu, etc) et outils d'installation (NSIS, nullsoft, Inno Setup). Utilisation autonome ou avec CMake, même syntaxe que CPack. Pour installation ou mise à jour (logiciel dédié ?) Qt Install Framework ?

```
// Dans CPackConfig.cmake
SET (CPACK_PACKAGE_NAME "compute_pi")
SET (CPACK_PACKAGE_VERSION "1.0")
SET (CPACK_PACKAGE_DESCRIPTION_SUMMARY "Parallel computation
of Pi")
SET (CPACK_PACKAGE_VENDOR "Sekou @ free.fr")
SET (CPACK_PACKAGE_DESCRIPTION_FILE
"${CMAKE_CURRENT_SOURCE_DIR}/README")
SET (CPACK_RESSOURCE_FILE_LICENCE
"${CMAKE_CURRENT_SOURCE_DIR}/Copyright.txt")
INSTALL (TARGETS computePi RUNTIME DESTINATION bin)

SET (CPACK_NSIS_HELP_LINK "http://devlper...")
SET (CPACK_DEBIAN_PACKAGE_DEPENDS "libglib-2.0-0 (>= 2.1)")
SET (CPACK_RPM_PACKAGE_REQUIRES "glib >= 2.1")

IF (WIN32)
    SET (CPACK_GENERATOR "ZIP; NSIS")
ELSE (WIN32)
    IF (APPLE)
        SET (CPACK_GENERATOR "TGZ")
    ELSE (APPLE)
        SET (CPACK_GENERATOR "TGZ; RPM; DEB")
    ENDIF (APPLE)
ENDIF (WIN32)

// composants
INSTALL (FILES ./include/compute_pi.h DESTINATION include
COMPONENT headers)
INSTALL (TARGETS compute_pi ARCHIVE DESTINATION lib
COMPONENT libraries)
INSTALL (TARGETS compute_pi DESTINATION bin COMPONENT
applications)
INSTALL (TARGETS computePi RUNTIME DESTINATION bin COMPONENT
applications
SET (CPACK_COMPONENT_APPLICATIONS_GROUP "Runtime")
SET (CPACK_COMPONENT_LIBRARIES_GROUP "Development")
SET (CPACK_COMPONENT_HEADERS_GROUP "Development")
SET (CPACK_COMPONENTS_ALL applications libraries headers)
INCLUDE (CPack)
```



Installateur généré avec CPack

Le suivi des problèmes rapportés par les utilisateurs.

bug tracker