INRIA

CONCOURS EXTERNE D'INGENIEUR DE RECHERCHE

ID1 / ID2

BAP E – INFORMATIQUE ET CALCUL SCIENTIFIQUE

Ouvert au titre de 2011
Arrêtés du 4 avril 2011 parus au JO du 12 avril 2011

Epreuve écrite
Note sur 20 – Coefficient 3 – Durée trois heures

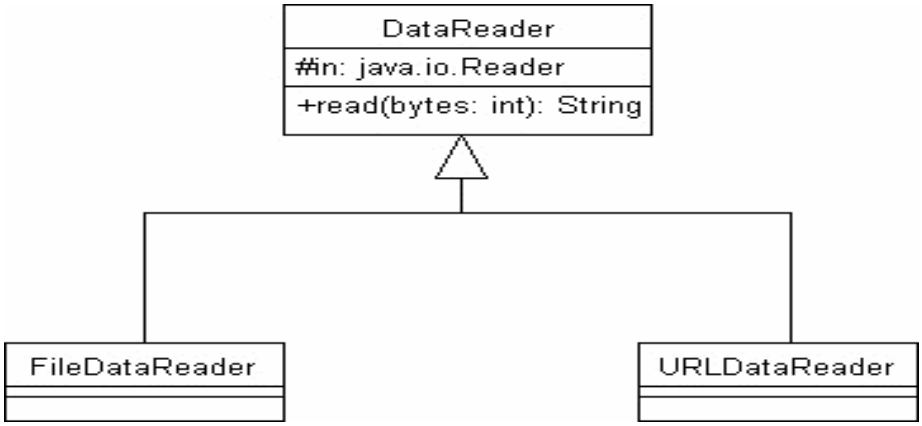
le 14 juin 2011 de 14h à 17h

Le candidat peut traiter les exercices dans l'ordre de son choix.
Veillez à respecter l'anonymat dans les réponses.

Module A : tronc commun aux concours ID1 & ID2 (total : 13 points)

Exercice A1 : (4 points)

On souhaite développer une application qui ait un comportement adapté aux besoins de l'utilisateur au cours de l'exécution du programme. Dans cet exemple élémentaire, deux types de lecteur de données pourront être utilisés en passant une chaîne de caractère en paramètre : `FileDataReader` (identifié par le paramètre "`File`") et `URLDataReader` (identifié par le paramètre "`Url`"). L'un et l'autre sont des héritages de la classe `DataReader` comme le montre le diagramme ci-dessous.



1. De façon générale, mentionnez l'intérêt d'utiliser les *design patterns*.
2. Donnez les éléments caractéristiques utilisés pour décrire un *design pattern*.
3. Présentez schématiquement et décrivez en 10 lignes maximum le *design pattern* à utiliser dans le problème mentionné.
4. Fournissez une implémentation minimale C++ pour répondre au problème (sans implémenter les classes `FileDataReader` ni `URLDataReader` dans le détail, l'accès aux données n'étant pas l'intérêt de la question).

Exercice A2 : (4 points)

Les systèmes décentralisés de contrôle de version (e.g. git, bazaar, mercurial) sont aujourd'hui couramment utilisés. Ces outils offrent un grand nombre de nouvelles possibilités d'organisation pour les projets de développement, notamment en permettant de s'adapter aux différentes situations et contraintes pour un projet donné.

1. Quels sont les principaux avantages d'utiliser un système de gestion de code source décentralisé?

2. Décrivez brièvement trois schémas d'organisation possibles pour une équipe de développement, grâce à un outil décentralisé.

3. Vous êtes responsable de l'organisation d'un projet qui développe une plate-forme logicielle dans laquelle plusieurs équipes de recherche participent. Il y a un certain nombre de modules de base qui seront distribués sous licence BSD. Chacune de ces équipes peut participer à l'élaboration ou à l'extension des modules de base et fournir de nouveaux modules qu'elle souhaite ou non rendre publics. Quel schéma d'organisation suggérez-vous pour les développeurs du projet ? Identifiez les rôles parmi ces développeurs.

Exercice A3 : (5 points)

Attention : pour cet exercice, vous devez utiliser les 4 documents de l'annexe

Les seuls documents à votre disposition sont ceux fournis dans l'annexe.

Attention de toujours respecter l'anonymat de votre copie

Vous travaillez dans un projet européen dans lequel les différentes équipes vont être amenées à proposer des choix techniques pour les développements qui seront faits en commun ou séparément. Les développements sont en C, C++ et Fortran.

La question de quel compilateur à utiliser est soulevée, sachant que les logiciels développés doivent fonctionner sur les plates-formes Linux, Windows et Mac. Vous devez rédiger un document synthétique (400 mots maximum) ***en anglais*** qui explique les avantages et les inconvénients de *gcc* et de *icc* aux responsables d'équipe.

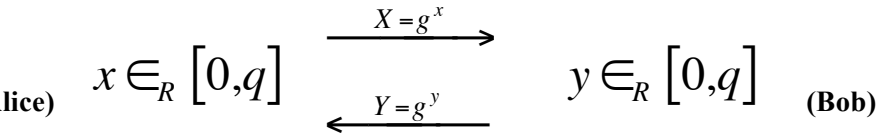
Module B : concours ID1 *uniquement* (total : 7 points)

Exercice B1 : (2 points)

1. Expliquez (en 10 lignes maximum) la différence entre un chiffrement symétrique et un chiffrement asymétrique. Citez 2 algorithmes de chaque type.
2. Dans la sécurité informatique, pour quelles raisons les protocoles de hachage sont-ils utilisés ? Pour assurer quelles propriétés ? Citez 2 protocoles de hachage.
3. Il existe plusieurs modèles d'authentification dans les réseaux 802.11. Listez-les et, pour chacun d'eux, indiquez succinctement la façon dont est créé le tunnel par lequel transitera le mot de passe.
4. Quelles attaques sont utilisées pour simuler une machine et s'emparer du login et du mot de passe des utilisateurs ?
5. Qu'est-ce qu'un protocole de signature aveugle ? Citez deux exemples d'applications qui s'appuient sur un tel protocole.

Exercice B2 : (2 points)

Alice et Bob veulent se mettre d'accord sur une clé commune et utilisent pour cela le protocole d'échange de clés de Diffie-Hellman :



Le nombre *q* est premier et *g* est un générateur du groupe multiplicatif des entiers modulo *q*.

1. Quelle est la valeur de la clé commune ainsi échangée ?
2. Décrire l'attaque appelée *man in the middle*.

Exercice B3 : (1 point)

Votre administrateur réseau découvre et vous informe qu'un utilisateur a réussi à obtenir des informations sur le serveur `asterix` grâce à la commande suivante :

```
telnet asterix.france.com 25
```

1. Pourquoi a-t-il réussi à obtenir des informations sur le serveur ?
2. Quelles informations peut-il obtenir ?
3. Quelle(s) contre-mesure(s) faut-il appliquer si l'utilisateur se situe à l'extérieur de l'entreprise ?

Exercice B4 : (1 point)

Alice vient vous voir pour échanger des données confidentielles avec Bob.

1. Quel procédé cryptographique lui conseillez-vous, un procédé à clé publique ou à clé privée ?

2. Quelle taille de clé recommandez-vous, en chiffrement à clé privée et en chiffrement à clé publique ? Justifiez votre réponse.

3. Alice veut envoyer un message à Bob en s'assurant : (1) que le message ne peut pas être lu par un intrus, (2) que Bob peut vérifier que le message provient bien d'Alice. Quel procédé proposez-vous ? Détaillez.

Exercice B5 : (1 point)

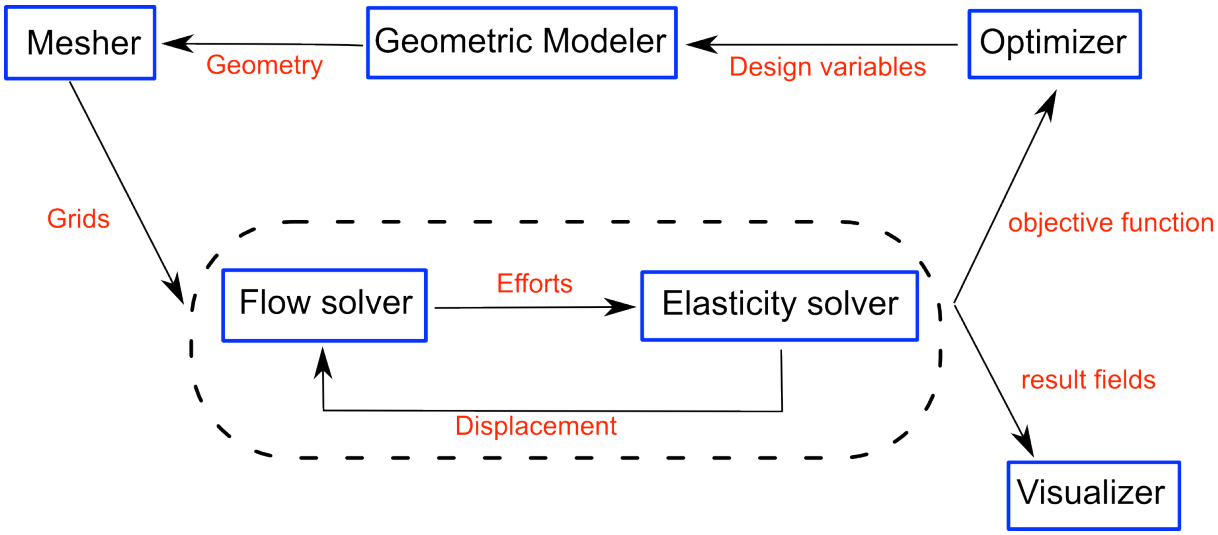
On accorde aux virus informatiques la capacité de chiffrement, de polymorphisme ou de métamorphisme. Décrivez en quelques lignes à quoi correspondent ces capacités.

Module C : concours ID2 *uniquement* (total : 7 points)

On cherche à optimiser la conception d'une voilure d'avion (interaction structure / fluide).

Les étapes nécessaires pour résoudre un tel problème d'optimisation sont les suivantes (voir aussi le schéma global, itératif, plus bas) :

- modeleur géométrique ;
- maillage ;
- solveur fluide-structure ;
- optimiseur ;
- visualisation.



Exercice C1 : (1 point)

On considère, pour le solveur structural, un modèle d'élasticité linéaire, conduisant à la résolution d'un système linéaire de type K.U = f.

1. Indiquez au moins 3 méthodes de résolution de systèmes linéaires. Dans le cas où K est une matrice creuse, indiquez la méthode et la bibliothèque logicielle que vous préconisez et justifiez votre réponse

2. Indiquez au moins 2 bibliothèques logicielles fournissant des solveurs linéaires

Exercice C2 : (1 point)

Pour le solveur fluide, on considère un modèle d'écoulements visqueux turbulents.

1. On envisage 2 approches pour répondre à ce problème :

- solveur implicite ;
- solveur explicite.

Expliquez les différences fondamentales entre ces 2 stratégies et les avantages et inconvénients de chacune d'elles dans le contexte présenté.

2. Par ailleurs, le couplage avec le solveur d'élasticité linéaire peut être conçu à l'aide d'un couplage fort ou un couplage faible. Expliquez ces termes et donnez les avantages et les inconvénients de chacune de ces stratégies de couplage dans le contexte présenté.

Exercice C3 : (2½ points)

Pour l'optimisation de la forme de la voilure, on va chercher à optimiser des critères de performance de cette dernière (portance, poids, ...).

Deux classes de méthodes d'optimisation sont pressenties : d'une part en calculant et en utilisant le gradient de la fonction objectif (méthode analytique), d'autre part avec une méthode heuristique (un algorithme randomisé, une méthode semi stochastique).

1. Exposez en quelques lignes les avantages et inconvénients de chaque approche et citez pour ces 2 classes de méthodes au moins 2 méthodes spécifiques.

2. En cas d'optimisation avec contraintes égalités, citez une méthode d'optimisation adaptée et un algorithme pour les mettre en œuvre.

Exercice C4 : (2½ points)

Pour des raisons de performance, on envisage de développer une version parallèle du code.

1. Sur ce cas, le parallélisme peut être fait à 2 niveaux algorithmiques différents : lesquels ?

2. Pour chaque niveau, donnez :

- le type de parallélisme / distribution à privilégier (quelles architectures cibles typiquement) et pourquoi ;
- les bibliothèques ou applications logicielles utiles au déploiement de l'application parallèle.

GNU Compiler Collection

From Wikipedia, the free encyclopedia

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain. As well as being the official compiler of the unfinished GNU operating system, GCC has been adopted as the standard compiler by most other modern Unix-like computer operating systems, including Linux, the BSD family and Mac OS X.[citation needed]

GCC has been ported to a wide variety of processor architectures, and is widely deployed as a tool in commercial, proprietary and closed source software development environments. GCC is also available for most embedded platforms, for example Symbian (called gcce),[2] AMCC and Freescale Power Architecture-based chips.[3] The compiler can target a wide variety of platforms, including videogame consoles such as the PlayStation 2[4] and Dreamcast.[5] Several companies[6] make a business out of supplying and supporting GCC ports to various platforms, and chip manufacturers today consider a GCC port almost essential to the success of an architecture.

Originally named the GNU C Compiler, because it only handled the C programming language, GCC 1.0 was released in 1987, and the compiler was extended to compile C++ in December of that year.[1] Front ends were later developed for Fortran, Pascal, Objective-C, Java, and Ada, among others.[7]

The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example.
[…]

Development

GCC stable release

The current stable version of GCC is 4.6.0, which was released on March 25, 2011. GCC 4.6 supports many new Objective-C features, such as declared and synthesized properties, dot syntax, fast enumeration, optional protocol methods, method/protocol/class attributes, class extensions and a new GNU Objective-C runtime API. It also supports the Go programming language and includes the libquadmath library, which provides quadruple-precision mathematical functions on targets supporting the __float128 datatype. The library is used to provide the REAL(16) type in GNU Fortran on such targets.

The previous major version, 4.5, was initially released on April 14, 2010 (last minor version is 4.5.2, released on December 16, 2010). It included several minor new features (new targets, new language dialects) and a couple major new features:
[…]

GCC trunk

The trunk concentrates the major part of the development efforts, where new features are implemented and tested. Eventually, the code from the trunk will become the next major release of GCC, with version 4.7.

Uses

GCC is often chosen for developing software that is required to execute on a wide variety of hardware and/or operating systems.[*citation needed*] System-specific compilers provided by hardware or OS vendors can differ substantially, complicating both the software's source code and the scripts which invoke the compiler to build it.[*citation needed*] With GCC, most of the compiler is the same on every platform, so only code which explicitly uses platform-specific features must be rewritten for each system.[*citation needed*]

Languages

The standard compiler release 4.6 includes front ends for C (gcc), C++ (g++), Java (gcj), Ada (GNAT), Objective-C (gobjc), Objective-C++ (gobjc++) and Fortran (gfortran).[16] Also available, but not in standard are Go (gccgo), Modula-2, Modula-3, Pascal (gpc), PL/I, D (gdc), Mercury, and VHDL (ghdl).[17] A popular parallel language extension, OpenMP, is also supported.

The Fortran front end was g77 before version 4.0, which only supports FORTRAN 77. In newer versions, g77 is dropped in favor of the new gfortran front end that supports Fortran 95 and parts of Fortran 2003 as well.[18] As the later Fortran standards incorporate the F77 standard, standards-compliant F77 code is also standards-compliant F90/95 code, and so can be compiled without trouble in gfortran. A front-end for CHILL was dropped due to a lack of maintenance.[19]

A few experimental branches exist to support additional languages, such as the GCC UPC compiler[20] for Unified Parallel C.

Architectures

GCC target processor families as of version 4.3 include:

Alpha	MIPS
ARM	PA-RISC
Atmel AVR	PDP-11
Blackfin	PowerPC
HC	R8C/M16C/M32C
H8/300	SPU
IA-32 (x86)	System/390/zSeries
x86-64	SuperH
IA-64	SPARC
Motorola 68000	VAX

[…]

Structure

GCC's external interface is generally standard for a UNIX compiler. Users invoke a driver program named gcc, which interprets command arguments, decides which language compilers to use for each input file, runs the assembler on their output, and then possibly runs the linker to produce a complete executable binary.

Each of the language compilers is a separate program that inputs source code and outputs assembly code. All have a common internal structure. A per-language front end parses the source code in that language and produces an abstract syntax tree ("tree" for short).

These are, if necessary, converted to the middle-end's input representation, called *GENERIC* form; the middle-end then gradually transforms the program towards its final form. Compiler optimizations and static code analysis techniques (such as FORTIFY_SOURCE,[24] a compiler directive which attempts to discover some buffer overflows) are applied to the code. These work on multiple representations, mostly the architecture-independent GIMPLE representation and the architecture-dependent RTL representation. Finally, assembly language is produced using architecture-specific pattern matching originally based on an algorithm of Jack Davidson and Chris Fraser.

GCC is written primarily in C except for parts of the Ada front end. The distribution includes the standard libraries for Ada, C++, and Java whose code is mostly written in those languages.[25] On some platforms, the distribution also includes a low-level runtime library, libgcc, written in a combination of machine-independent C and processor-specific assembly language, designed primarily to handle arithmetic operations that the target processor cannot perform directly.[26]

In May 2010, the GCC steering committee decided to allow use of a C++ compiler to compile GCC.[27] The compiler will be written in C plus a subset of features from C++. In particular, this was decided so that GCC's developers could use the "destructors" and "generics" features of C++.[28]

[…]

Optimization

Optimization can occur during any phase of compilation, however the bulk of optimizations are performed after the syntax and semantic analysis of the front-end and before the code generation of the back-end, thus a common, even though somewhat contradictory, name for this part of the compiler is "middle end."

The exact set of GCC optimizations varies from release to release as it develops, but includes the standard algorithms, such as loop optimization, jump threading, common subexpression elimination, instruction scheduling, and so forth. The RTL optimizations are of less importance with the addition of global SSA-based optimizations on GIMPLE

trees,[33] as RTL optimizations have a much more limited scope, and have less high-level information.

Some of these optimizations performed at this level include dead code elimination, partial redundancy elimination, global value numbering, sparse conditional constant propagation, and scalar replacement of aggregates. Array dependence based optimizations such as automatic vectorization and automatic parallelization are also performed. Profile-guided optimization is also possible as demonstrated here: http://gcc.gnu.org/install/build.html#TOC4

[…]

Debugging GCC programs

The primary tool used to debug GCC code is the GNU Debugger (gdb). Among more specialized tools are Valgrind, for finding memory errors and leaks, and the graph profiler (gprof) which can determine how much time is spent in which routines, and how often they are called; this requires programs to be compiled with profiling options.

Intel C++ Compiler

From Wikipedia, the free encyclopedia

Intel C++ Compiler (also known as **icc** or **icl**) is a group of C and C++ compilers from Intel Corporation available for GNU/Linux, Mac OS X, and Microsoft Windows.

Intel supports compilation for its IA-32 and Intel 64 processors and certain non-Intel but compatible processors, such as certain AMD processors. Developers should check system requirements. The Intel C++ Compiler for IA-32 and Intel 64 features an automatic vectorizer that can generate SSE, SSE2, SSE3 and SSE4 SIMD instructions, the embedded variant for Intel Wireless MMX and MMX 2.[1] Since its introduction, the Intel C++ Compiler for IA-32 has greatly increased adoption of SSE2 in Windows application development.[*citation needed*]

Intel C++ Compiler further supports both OpenMP 3.0 and automatic parallelization for symmetric multiprocessing. With the add-on capability Cluster OpenMP, the compiler can also automatically generate Message Passing Interface calls for distributed memory multiprocessing from OpenMP directives.
Intel C++ Compiler belongs to the family of compilers with the Edison Design Group frontend (like the SGI MIPSpro, Comeau C++, Portland Group, and others). The compiler is also notable for being widely used for SPEC CPU Benchmarks of IA-32, x86-64, and Itanium 2 architectures.

The Intel C++ Compiler is available in four forms. It is part of Intel Parallel Studio, the Intel C++ Compiler Professional Edition package, the Intel Compiler Suite package and the Intel Cluster Toolkit, Compiler Edition. The Intel Software Products site provides more information.
[…]

Optimizations

Intel tunes its compilers to optimize for its hardware platforms to minimize stalls and to produce code that executes in the fewest number of cycles. The Intel C++ Compiler supports three separate high-level techniques for optimizing the compiled program: interprocedural optimization (IPO), profile-guided optimization (PGO),[2] and high-level optimizations (HLO). It also supports tools and techniques for adding and maintaining parallelism to applications.

Profile-guided optimization refers to a mode of optimization where the compiler is able to access data from a sample run of the program across a representative input set. The data would indicate which areas of the program are executed more frequently, and which areas are executed less frequently. All optimizations benefit from profile-guided feedback because they are less reliant on heuristics when making compilation decisions.

High-level optimizations are optimizations performed on a version of the program that more closely represents the source code. This includes loop interchange, loop fusion,

loop unrolling, loop distribution, data prefetch, and more.[3] These optimizations are usually very aggressive and may take considerable compilation time.

Interprocedural optimization applies typical compiler optimizations (such as constant propagation) but using a broader scope that may include multiple procedures, multiple files, or the entire program.[4]

The compilers include a parallel debugger extension, Intel Threading Building Blocks, lambda function support, and a source checker tool for use with threaded code.

Intel's compiler has been criticized for applying, by default, floating-point optimizations not allowed by the C standard and that require special flags with other compilers such as gcc.[5]

Languages

Intel's suite of compilers has front ends for C, C++, and Fortran.

Early versions of ICC for Linux that predate GCC 3.x use the Dinkumware name mangling scheme in order to provide a more standard implementation of C++ than GCC 2.x. This made its ABI incompatible with both GCC versions. Intel removed the Dinkumware libraries in the 10.0 release (June 2007). Since then, the compiler has been and remains compatible with GCC 3.2 and later.

Architectures

- IA-32
- x86-64 (Intel 64 and AMD64)

Versions

The following versions of Intel C++ Compiler have been released:

Compiler version	Release date	Major New Features
Intel C++ Compiler XE 12.0	Nov 7, 2010	Intel Cilk Plus language extensions, Guided Auto-Parallelism, Improved C++0x support.[6]
Intel C++ Compiler 11.1	June 23, 2009	Support for latest Intel SSE SSE4.2, AVX and AES instructions. Parallel Debugger Extension. Improved integration into Microsoft Visual Studio, Eclipse CDT 5.0 and Mac Xcode IDE.
Intel C++ Compiler 11.0	November 2008	Initial C++0x support [1]. VS2008 IDE integration on Windows. OpenMP 3.0. Source Checker for static memory/parallel diagnostics.
[…]	[…]	[…]

[…]

Flags and manuals

Documentation can be found at the Intel Software Technical Documentation site.

Windows	Linux & MacOSX	Comment
/Od	-O0	No optimization
/O1	-O1	Optimize for size
/O2	-O2	Optimize for speed and enable some optimization
/O3	-O3	Enable all optimizations as O2, and intensive loop optimizations
/QxO	-xO	Enables SSE3, SSE2 and SSE instruction sets optimizations for non-Intel CPUs [17]
/fast	-fast	Shorthand. On Windows this equates to "/O3 /Qipo /QxHost /no-prec-div" ; on Linux "-O3 -ipo -static -xHOST -no-prec-div". Note that the processor specific optimization flag (-xHOST) will optimize for the processor compiled on—it is the only flag of -fast, which may be overridden.
/Qprof-gen	-prof_gen	Compile the program and instrument it for a profile generating run.
/Qprof-use	-prof_use	May only be used after running a program that was previously compiled using prof_gen. Uses profile information during each step of the compilation process.

Debugging

The Intel compiler provides debugging information that is standard for the common debuggers (DWARF 2 on Linux, similar to gdb, and COFF for Windows). The flags to compile with debugging information are /Zi on Windows and -g on Linux.

Intel also provides its own debugger called *idb*, which can be run in both dbx and gdb compatible command mode.

While the Intel compiler can generate a gprof compatible profiling output, Intel also provides a kernel level, system-wide statistical profiler as a separate product called VTune. VTune features an easy-to-use GUI (integrated into Visual Studio for Windows, Eclipse for Linux) as well as a command line interface.
The 11.x releases of the compiler introduced the Parallel Debugger Extension, which provides techniques for debugging threaded applications. It can be used with other, compatible compilers, such as Microsoft Visual C++ on Windows as available in Visual Studio 2005 and 2008 and gcc on Linux.

Criticism

Intel has published benchmarks to substantiate performance leadership claims over open source and AMD libraries on Intel and non-Intel processors. Intel and AMD have documented flags to use on the Intel compilers to get optimal performance on Intel and AMD processors.[18][19] Nevertheless, the Intel compilers have been accused of producing sub-optimal code. Here is an almost-verbatim quote from a blog[20]: The Intel compiler and several different Intel function libraries have suboptimal performance on AMD and VIA processors. The reason is that the compiler or library can make multiple versions of a piece of code, each optimized for a certain processor and instruction set, for example SSE2, SSE3, etc. The system includes a function that detects which type of CPU it is running on and chooses the optimal code path for that CPU. This is called a CPU dispatcher. However, the Intel CPU dispatcher does not only check which instruction set is supported by the CPU, it also checks the vendor ID string. If the vendor string is "GenuineIntel" then it uses the optimal code path. If the CPU is not from Intel then, in most cases, it will run the slowest possible version of the code, even if the CPU is fully compatible with a better version.

This vendor-specific CPU dispatching decreases the performance on non-Intel processors of software built with an Intel compiler or an Intel function library - possibly without the knowledge of the programmer. This has allegedly led to misleading benchmarks.[20] A legal battle between AMD and Intel over this and other issues has been settled in November 2009.[21] In late 2010, Intel settled an US Federal Trade Commission antitrust investigation against Intel.[22]

ANNEXE: document #3

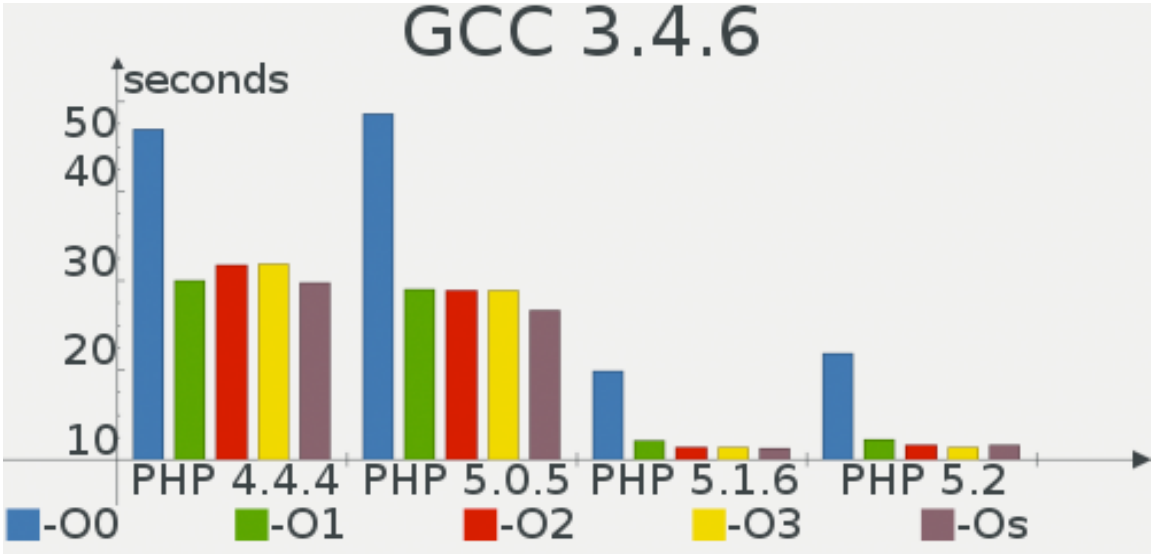
Extrait de http://sebastian-bergmann.de/archives/634-PHP-GCC-ICC-Benchmark.html

PHP / GCC / ICC Benchmark

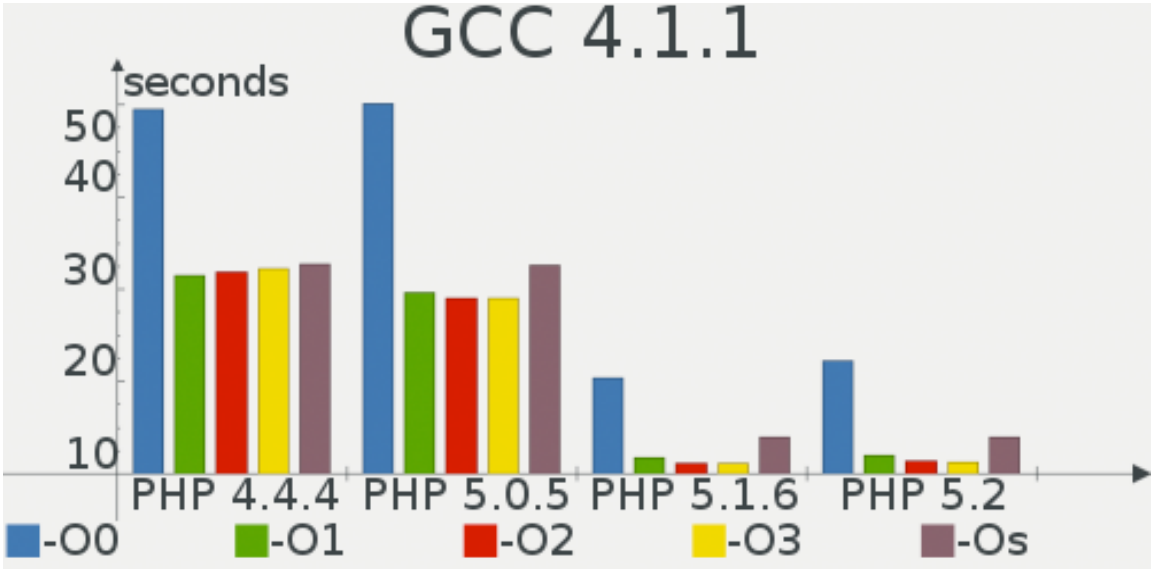
Sebastian Bergmann » 07 November 2006 » in Benchmarks » 4 Comments

Last month I got a bit bored and built PHP 4.4.4, PHP 5.0.5, PHP 5.1.6, and current PHP_5_2 (the last two each with CALL, GOTO, and SWITCH VMs) with GCC 3.4.6 and GCC 4.1.1 with -O{0|1|2|3|s}. Yes, this means I built 80 PHP binaries. No, I did not do this manually.

Below are the results of running bench.php with each of the binaries that I built.



As you can see, PHP 5.1 and PHP 5.2 are both around three times faster than PHP 4.4 and PHP 5.0 (GCC 3.4.6, -O2).

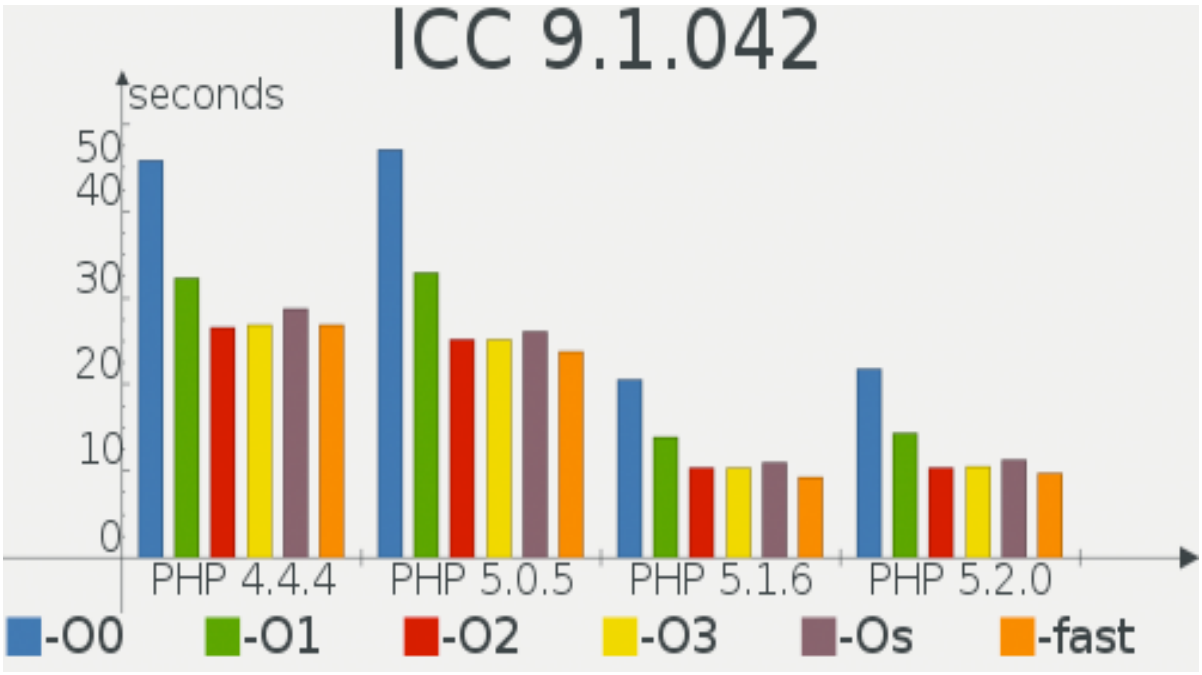


Each binary ran the benchmark script five times, the numbers shown are the respective mean from the five runs.

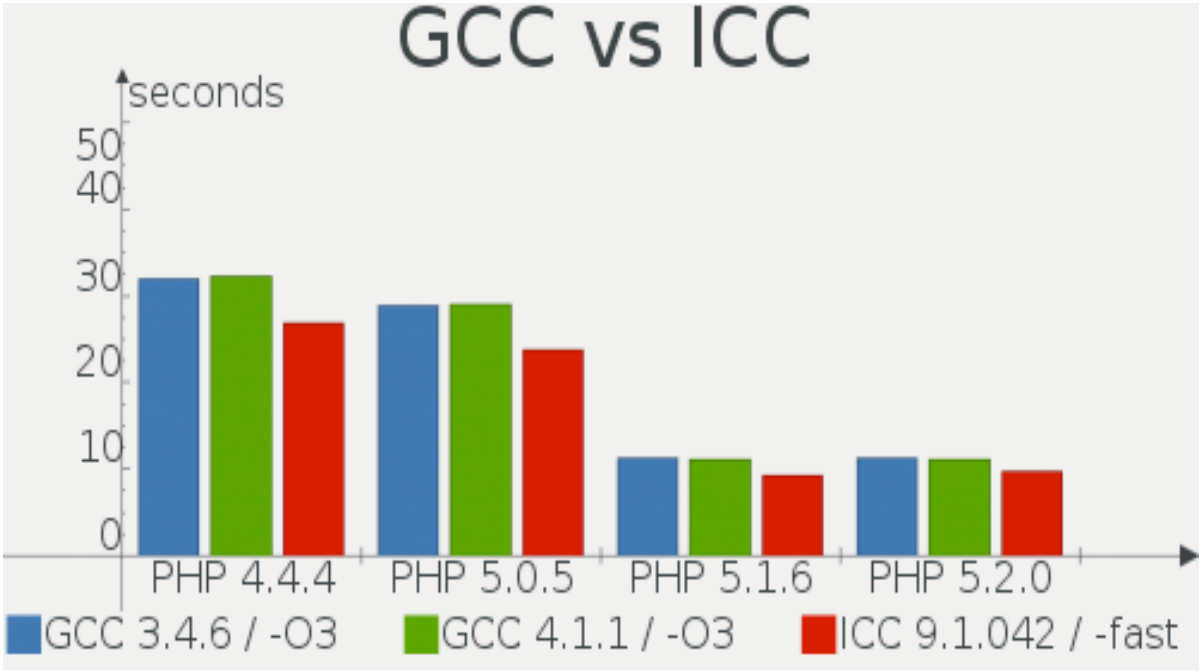
The numbers for PHP 5.1.6 and PHP 5.2 are based upon the CALL-based virtual machine as GOTO and SWITCH did not work for most of the optimization levels.

ANNEXE: document #3

Update: Yesterday I built PHP 4.4.4, PHP 5.0.5, PHP 5.1.6, and PHP 5.2.0 with ICC 9.1.042 using -O{0|1|2|3|s} and -fast.



Below is a chart that directly compares GCC 3.4.6, GCC 4.1.1, and ICC 9.1.042.



The GCC versions used were *Gentoo sys-devel/gcc-4.1.1-r1* and *Gentoo sys-devel/gcc-3.4.6-r2* together with *Gentoo sys-devel/binutils-2.17*.
The CFLAGS used for GCC were "-march=pentium-m -msse3 -O{0|1|2|3|s} -pipe".

The configure options used were "--disable-all --disable-cgi".

The extended body of this posting contains the raw test results as well as detailed information on the hardware used to run the benchmarks.

ANNEXE: document #4

Extract de
http://news.povray.org/povray.general/thread/%3C4118a9bc$1@news.povray.org%3E/

BENCHMARK: Itanium, Opteron, Xeon, Athlon (GCC, ICC)

Subject: BENCHMARK: Itanium, Opteron, Xeon, Athlon (GCC, ICC)
Date: 10 Aug 2004 10:55:56

Hi!
I have had the chance to compile and run POV3.6 on various platforms using Gnu C/C++ Compiler and Intel C/C++ Compiler.
I have run tests on these Linux-Machines (further specs see below):
* INTEL Itanium, 0.8 GHz,
* AMD Athlon TB, 0.8 GHz,
* AMD Opteron 244, 1.8 GHz,
* INTEL Pentium IV Xeon, 2.4 GHz

Of course, it's not particularly fair to compare a 1.8GHz Opteron machine with a 0.8 GHz Athlon --keep that in mind, please!

Benchmark for var. archs and compilers:
using POV-Rays official benchmark.pov script
with suggested options -w384 -h384 +a0.3 +v -d -f -x

numbers = processor time [seconds]
LOWER NUMBERS = BETTER

	ICC8.0	GCC3.x.x
ITANIUM (64)	10880	8715
OPTERON (64)	n.a.	1580
XEON (32)	2458	3757
ATHLON (32)	n.a.	5012

(64=64bit arch, 32=32bit arch)

* AMD Opteron 244, 1.8 GHz, 4 GByte RAM
  (dual processor boards in cluster) RedHat Linux
* INTEL Pentium IV Xeon, 2.4 GHz, 2 GByte RAM
  (dual processor boards in cluster) RedHat Linux
* INTEL Itanium, 0.8 GHz, 2 GByte Ram
  (dual processor board) Debian Linux
* AMD Athlon TB, 0.8 GHz, 256 kB Ram
  (single processor board) Debian Linux

Obviously, you won't make use of an Itanium machine because of it's speed ;-)
I was wondering why the Intel Compiler produced a code that performed ~20% slower than the one produced by GCC, so I randomly chose some of POVs example-scene files for render with both codes, and here is what I got:

SCRIPT	ICC8.0	GCC3.x
box.pov	5.87	5.80
glassthing.pov	299.09	320.25
parallel_lights.pov	19.37	21.69
circular.pov	143.74	175.14
shadows.pov	67.86	89.82
fog_ft.pov	10.30	9.21
atten2.pov	30.70	34.06
caustic2.pov	22.60	19.50
skysph2.pov	32.85	21.26
radiosity2.pov	187.19	207.37
cornell.pov	71.10	81.8
shear.pov	0.03	0.03
perspective.pov	0.02	0.03
panoramic.pov	0.03	0.03
focalblur.pov	0.03	0.03
abyss.pov	486.54	536.31
gaussianblob.pov	213.71	230.85
MEAN	91.00	99.70

So here is what I expected. Overall-performance codes produced by the Intel-Compiler is ~10% better than GCC-Codes. It seems, that calculation of simple objects with no further options like fog etc perform better with GCC-POV Code. However, scripts like 'circular.pov', 'abyss.pov' or 'shadows.pov' show drastical improovement with ICC-POV-Code.

I run the same scripts on the Xeon Machine and noticed an overall improovement of approx. 12% using ICC (ICC: 28.9s, GCC: 33.0s). Comments and suggestions welcome... Sincerely S. Tayefeh

Comments and suggestions welcome...
Sincerely S. Tayefeh

ITANIUM : GCC-OPTIONS:

```
CFLAGS =
CPP = gcc -E
CPPFLAGS =  -I/usr/X11R6/include
CXXCPP = g++ -E
CXXFLAGS =  -pipe -Wno-multichar -O3
LDFLAGS =  -L/usr/X11R6/lib
LIBS = -ltiff -ljpeg -lpng12 -lz -lXpm   -lSM -lICE -lX11 -lm
X_CFLAGS =  -I/usr/X11R6/include
X_LIBS =  -L/usr/X11R6/lib
X_PRE_LIBS =  -lSM –lICE
```

ITANIUM : ICC-options:

```
CFLAGS =  -g
CPP = /<path>/icc -E
CPPFLAGS =  -I/usr/X11R6/include
CXX = /<path>/icc
CXXCPP = /<path>/icc -E
CXXDEPMODE = depmode=gcc3
CXXFLAGS =  -O3
LDFLAGS =  -L/usr/X11R6/lib
LIBS = -ltiff -ljpeg -lpng12 -lz -lXpm   -lSM -lICE -lX11 -lm
```

```
X_CFLAGS =  -I/usr/X11R6/include
X_LIBS =  -L/usr/X11R6/lib
X_PRE_LIBS =  -lSM -lICE
```

XEON: GCC-OPTIONS:

CFLAGS = -pipe -O3 -msse -malign-double -minline-all-stringops
CPP = gcc -E
CPPFLAGS = -I/usr/X11R6/include
CXX = g++
CXXCPP = g++ -E
CXXDEPMODE = depmode=gcc3
CXXFLAGS = -pipe -Wno-multichar -O3 -msse -march=i686 -malign-double -minline-all-stringops
LDFLAGS = -L/usr/X11R6/lib
LIBS = -ljpeg -lXpm -lSM -lICE -lX11 -lm
X_CFLAGS = -I/usr/X11R6/include
X_EXTRA_LIBS =
X_LIBS = -L/usr/X11R6/lib
X_PRE_LIBS = -lSM -lICE

XEON: ICC-OPTIONS:

CFLAGS = -O3 -ip -march=pentium4 -mcpu=pentium4
CPP = icc -E
CPPFLAGS = -I/usr/X11R6/include
CXX = icc
CXXCPP = icc -E
CXXDEPMODE = depmode=icc
CXXFLAGS = -O3 -ip -march=pentium4 -mcpu=pentium4
LDFLAGS = -L/usr/X11R6/lib
LIBS = -ljpeg -lXpm -lSM -lICE -lX11 -lm
X_CFLAGS = -I/usr/X11R6/include
X_LIBS = -L/usr/X11R6/lib
X_PRE_LIBS = -lSM -lICE

OPTERON: GCC-Options:

CFLAGS = -pipe -O3 -msse -mfpmath=sse -msse2 -march=k8 -mcpu=k8 -minline-all-stringops
CPP = gcc -E
CPPFLAGS = -I/usr/X11R6/include
CXX = g++
CXXCPP = g++ -E
CXXFLAGS = -pipe -Wno-multichar -O3 -msse -mfpmath=sse -msse2 -march=k8 -mcpu=k8 -minline-all-stringops
LDFLAGS = -L/usr/X11R6/lib64
LIBS = -ljpeg -lXpm -lSM -lICE -lX11 -lm
X_CFLAGS = -I/usr/X11R6/include
X_LIBS = -L/usr/X11R6/lib64
X_PRE_LIBS = -lSM -lICE