Advanced Computer Systems and
Compiler Laboratory

WIPI 환경을 위한 Java-to-C 컴파일러

2004년도 프로그래밍언어 연구회 여름학교
2004. 8.11.

고려대학교 전자공학과
김선욱
http://compiler.korea.ac.kr

본 주제의 기초 연구는 2003년 전자통신연구원 용역과제로 수행되었으며, 확장 연구는 2004년 정보통신연구진흥원 학술용역과제로 수행되고 있음.

Contents

- 무선 인터넷 플랫폼
- Java-to-C compiler
- 향후 연구 방향

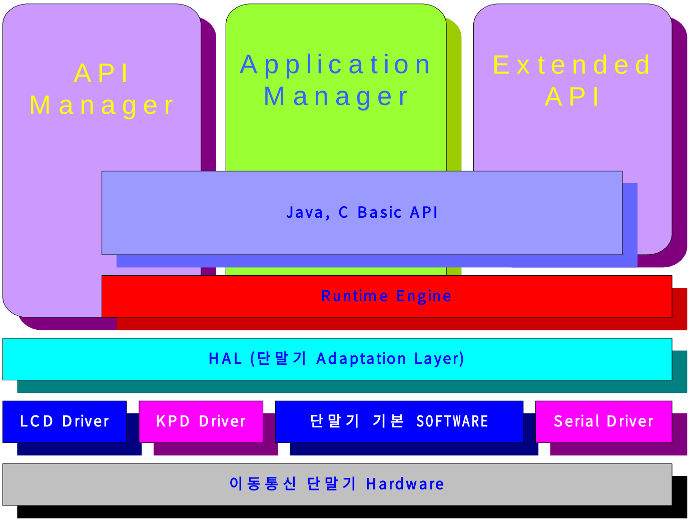
무선 인터넷 플랫폼

- WIPI
- BREW
- Sun JDK

WIPI (Wireless Internet Platform for Interoperability)

- 이동통신 단말기에 탑재되어 무선 인터넷을 통해 다운로드 된 응용프로그램 실행 환경을 제공하는데 필요한 표준 규격.
- 플랫폼과 어플리케이션은 하드웨어에 독립적인 구현이 가능하도록 설계.
- 어플리케이션이 정보를 마음대로 접근할 수 없도록 하는 보안규격 포함.
  - 이동통신 사업자 및 단말기 제조사 비밀.
  - 단말기 사용자 개인정보.
- 지원언어는 C, Java.

WIPI 플랫폼의 전체적인 시스템 구조



API Manager

Application Manager

Extended API

Java, C Basic API

Runtime Engine

HAL (단말기 Adaptation Layer)

LCD Driver

KPD Driver

단말기 기본 SOFTWARE

Serial Driver

이동통신 단말기 Hardware

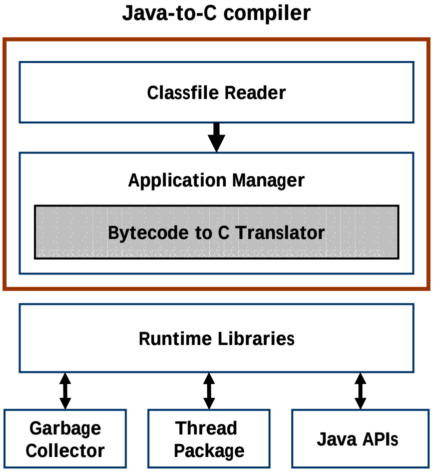
Performance Issues on WIPI

- Support interoperability!
  - Several layers, i.e. HAL.
  - Supporting Java.
    - □ JVM
    - □ JIT (Just-In-Time) compiler.
    - □ Java processor.
    - □ AOT (Ahead-of-Time) compiler
      - □ Java-to-C compiler.

Java-to-C Translator

- Building a Java-to-C (Java2C) compiler that is still preserving Java semantics
  - □ Inheritance
  - □ Method overloading
  - □ Virtual method invocation
- Fully supports CLDC (Connected Limited Device Configuration) 1.0 API.
- Performance improvement in terms of execution time and code size over the other methods.

Overall Structure



Java-to-C compiler

Classfile Reader

Application Manager

Bytecode to C Translator

Runtime Libraries

Garbage Collector

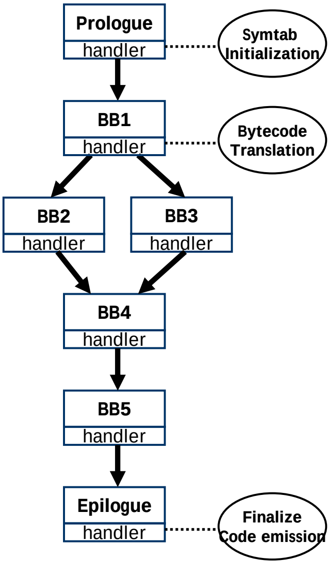
Thread Package

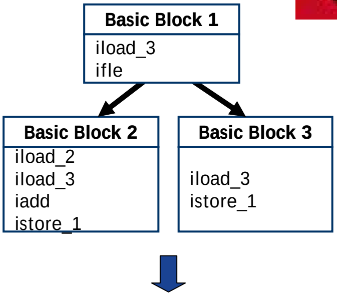
Java APIs

Components

- Classfile reader
  - Each classfile is translated into class blocks.
  - Class blocks to maintain information such as fields, methods, and a super class.
  - All associated class blocks are *also translated* together.
- Bytecode-to-C translator
  - Actual code generator
    - □ Build CFG.
    - □ Assign temporal variables by stack tracing.
    - □ C code generation.

Overview of bytecode-to-C translation process





In C code

```
BB1 :          i0 = Lvi3
               if( i0 <= 0 )    goto _L14;
BB2 :   _L10:  pc = 10;
               i0 = Lvi2;
               i1 = Lvi3;
               i0 = i0 + i1;
               Lvi1 = i0;
BB3 :   _L14:  pc = 14;
               i0 = Lvi3;
               Lvi1 = i0;
```

Components (cont)

- Application manager
  - Generates a complete C program
  - The manager has prototypes for Java runtime data structures, class initialization, method invocation, garbage collection and a main method to start up translated C programs.
- Runtime libraries
  - Native methods depend on a target system.
  - Currently fully supports CLDC 1.0 API.
  - Supports garbage collection and thread management.

Runtime Structures

- When our Java-to-C compiler translates a Java application into C codes, it needs such runtime structures to conserve Java's object-oriented features such as inheritance, method overloading and virtual method invocation.
- Considerations
  - Naming convention
  - Data layout
  - Method invocation

Naming Convention

- Java entities such as a class and a method are uniquely identified *by their names and an additional hash-code suffix* for avoiding any naming conflict in a global namespace of C program.
- The name of each Java method is also mapped to a different C name, and therefore an additional Java feature such as method overloading is naturally supported.

Data Layout

- Java primitive types are translated into primitive C types of the same data size.
  - Java's character type → an unsigned short type in C.
  - The reference type → a C pointer type.
    - Java objects and arrays are reference types that extend java.lang.Object class.
    - Each reference points to the runtime data structure for an object or an array in C.
- The data structure has a pointer to a common class structure which is constructed with the following three components:
  - a class descriptor table: contains general information needed for all classes
  - a methods table
  - a static variables table.

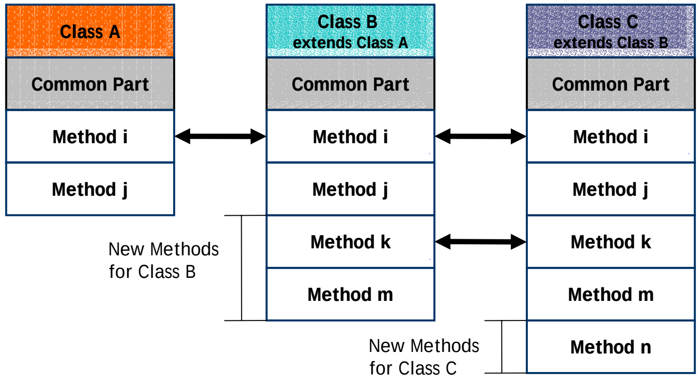
Class Descriptor Table

int need_init	The flag contains whether the class was already initialized or not.
int flag	Contains access information of the class.
int instance_size	The byte size of the class.
j_class super	Pointer to the parent class.
j_class array_class	Pointer to array class of the class.
j_class elem_class	Pointer to element class, if the class is array class.
ihash *method_hash	Contains hash codes for each method.
int method_num	The number of methods in the class.
void (*static_const_)()	Pointer to the static class initializer.
void (*def_const_)()	Pointer to the default constructor.
void (*finalize_)()	Pointer to the function for finalization.

Method Table

- Contains several function pointers to the invoked actual methods.
- During the Java-to-C compilation, the pointers are overwritten according to the inheritance relation.

Inheritance between Classes



Class A

Common Part

Method i

Method j

Class B
extends Class A

Common Part

Method i

Method j

Method k

Method m

New Methods
for Class B

Class C
extends Class B

Common Part

Method i

Method j

Method k

Method m

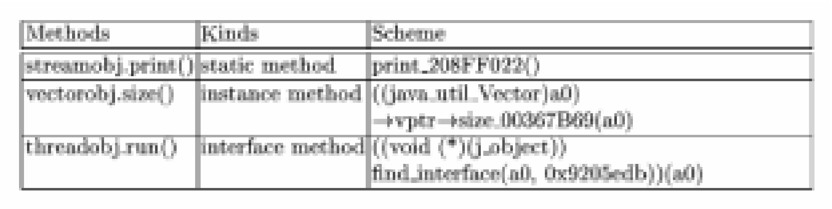
Method n

New Methods
for Class C

Method Invocation

- Java methods can be invoked in several ways according to how the methods are referenced.
- The static methods including constructors are always invoked without reference to a particular object but a class.
  - The methods should guarantee that the class includes pointers to themselves that has been already initialized before invocation.
- Instance methods are referenced by a specific object, and it is determined by runtime symbolic link.
- When an interface method is invoked, our Java-to-C compiler performs exhaustive search to find the method that will be invoked.

Method Invocation (cont)



Methods	Kinds	Scheme
streamobj.print()	static method	print_208FF022()
vectorobj.size()	instance method	((java_util_Vector)a0) →vptr→size_00367B69(a0)
threadobj.run()	interface method	((void (*)(j_object)) find_interface(a0, 0x9205edb))(a0)

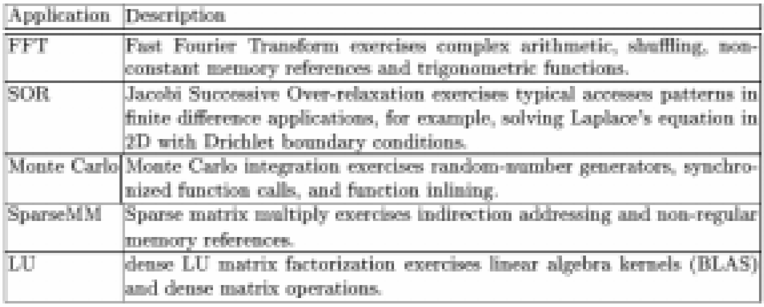
Etc.

- Exception handling
- Garbage collection
- Thread management

Performance evaluation

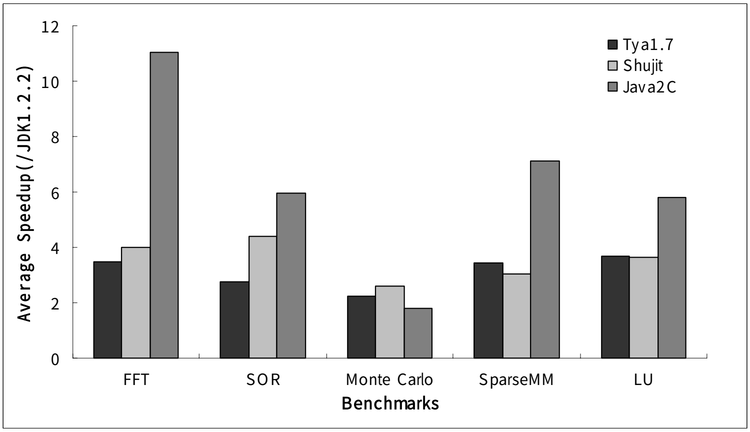
- Benchmark
  - Java SciMark 2.0
- Platforms
  - JDK 1.2.2.
  - Tya 1.7 JIT
  - ShuJIT.
- Machines
  - Zeon 2.0GHz
  - 512MB memory
  - Linux/Redhat 9.0

Java SciMark 2.0

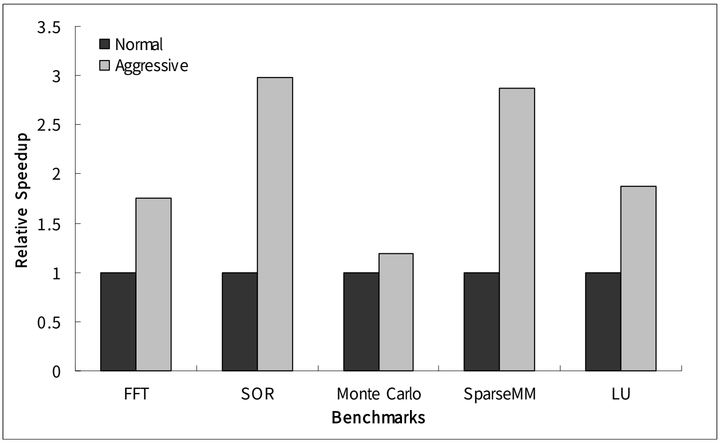


Application	Description
FFT	Fast Fourier Transform exercises complex arithmetic, shuffling, non-constant memory references and trigonometric functions.
SOR	Jacobi Successive Over-relaxation exercises typical accesses patterns in finite difference applications, for example, solving Laplace's equation in 2D with Drichlet boundary conditions.
Monte Carlo	Monte Carlo integration exercises random-number generators, synchronized function calls, and function inlining.
SparseMM	Sparse matrix multiply exercises indirection addressing and non-regular memory references.
LU	dense LU matrix factorization exercises linear algebra kernels (BLAS) and dense matrix operations.

Performance with Exception Handling



Performance without Exception Handling



추후 연구 방향

- Code size reduction
  - Interprocedural analysis.
- Code optimization
  - Many pointers prevent a backend compiler from optimizing the generated C codes.
- Interoperability with native methods
  - Different data layouts.

질의 응답