Portable Native Client

David Sehr, Robert Muth, Jan Young, David Meyer, Betul Buyukkurt, Karl Schimpf, Jason Kim, Rafael Espindola, Alan Donovan
Agenda

Motivation
Approach
Safe Translation
Bitcode as an Interchange Format
Status
Future Work
Motivation
A NaCl-Enabled Web Application

Your favorite language
A NaCl-Enabled Web Application

Your favorite language

Screened for malicious instructions
A NaCl-Enabled Web Application

Your favorite language

Screened for malicious instructions

System calls moderated by a virtualized OS
A NaCl-Enabled Web Application

Your favorite language

Screened for malicious instructions

System calls moderated by a virtualized OS

Performance within 5% of native code
Applications with NaCl

Lego Star Wars

Nexuiz

Darkroom demo

portlet to Native Client and to OpenGL ES 2.0.
Where Native Client Started

OSX
Windows
Linux

x86-32
Where We Went Next

ARM

x86-32
What Developers Want

Only one porting effort
Approach
Application Life Cycle

sources
- app.cc
- app.so
- lib.bc

library
info

A Global Cooling Game

Image Credit: NASA Goddard Space Flight Center Image by Reto Stöckli (land surface, shallow water, clouds)
Enhancements by Robert Simmon (ocean color, compositing, 3D globe, animation). Data and technical support: MODIS Land Group, MODIS Surface Data Support Team, MODIS Atmosphere Group, MODIS Ocean Group, Additional data: USGS EROS Data Center (topography), USGS Terrestrial Remote Sensing Flagstaff Field Center (Antarctica), Defense Meteorological Satellite Program (nightlight).
Bitcode is PNaCl’s distribution format
Client side

Native Client Helper

http://myurl/myapp.bc

translation engine

myapp.so

NaCl sandbox

ELF x86, x64, or ARM
Translation Engine

myapp.bc

in my cache?

translator (llvm)

save translation

use cached translation

myapp.so

Know the platform (uarch)
Translation Engine

myapp.bc

in my cache?

translator (llvm)

save translation

use cached translation

myapp.so

Know the platform (uarch)
Can collect/use profiling data
Webpage-specific specialization
Translation Engine

myapp.bc

in my cache?

translator (llvm)

save translation

use cached translation

myapp.so

Know the platform (uarch)
Can collect/use profiling data
Webpage-specific specialization
Can translate at invocation time
install time asynchronously
Safe Translation
Translating in a Sandbox

The translator must run in the browser
Malicious bitcode files are a potential attack vector
Translating in a Sandbox

The translator must run in the browser
Malicious bitcode files are a potential attack vector

Translator phases are run as NaCl modules
today

myapp.bc

llc

as

ld

myapp.so

what we want

myapp.bc

llc

myapp.so
Today

myapp.bc

llc

as

ld

myapp.so

What we want

myapp.bc

llc

myapp.so

+ MC ELF
+ Bundling
+ DT_NEEDED
Bitcode as an Interchange Format
Target Model

Address space / data model
  ILP32 (sizeof(int) == sizeof(long) == sizeof(void*))
  sizeof(va_list) == 24
  1GB maximum total address space
  Stack pointer starts at the top of the address space

Data types
  IEEE fp
  “natural” alignment
    (e.g., double is aligned 0mod8)

Byte order
  Little Endian
Target Model

C++ Exception Handling

x86-32 Linux model

varargs

sizeof(va_list) == 24

Front end emits va_arg instruction

setjmp

Consistent jmp_buf size (work in progress)
**Target Model**

**Calling conventions**
- Bitcode file is calling convention neutral
- Actual target convention determined by translator

**Concurrency and memory model**
- Assume a least common denominator
  - Store ordering within a thread
  - Explicit synchronization across threads

We expect people to use llvm atomic/barrier intrinsics where needed
Bitcode as an Interchange Format

PNaCl will need bitcode stability
   Developer expects published bitcode to work forever
Download size is startup time
   .bc is ~3x bigger than .nexe, ~1.9x when .gz
   .bc is ~6x bigger than .NET
How should we handle bitcode versioning?
Bitcode as an Interchange Format

PNaCl will need bitcode stability
  Developer expects published bitcode to work forever
Download size is startup time
  .bc is ~3x bigger than .nexe, ~1.9x when .gz
  .bc is ~6x bigger than .NET
How should we handle bitcode versioning?

We need your help!

Thursday, November 4, 2010
What’s running?

One bitcode file translates, validates, and runs on three architectures
  All of SPEC2000 int and the four C fp tests pass

The translator is sandboxed
  llc, as, ld runs as a NaCl module on x86-32 and 64

A few areas of portability work remain
  C++ exception handling on ARM is incomplete
  setjmp/longjmp is just coming together
CodeGen Work

Control and data sandboxing on ARM
  Robert, Cliff

Control and data sandboxing on x86
  Robert, Alan, Jan, David

ILP32 on x86-64
  Jan, David

x86-32 and x86-64 MC ELF contributions
  Rafael

ARM MC ELF contributions
  Jason
Front end work

ILP32 for x86-64
  Jan, DavidM

Varargs
  DavidM

Exception handling, setjmp
  Robert
Future Work
Directly Producing .so’s

ELF MC
   ARM support is still incomplete

MCAssambler
   “Bundling” support for NaCl pseudo-instructions

.so generation
   Simulated linking to collect symbols
   Emission work for DT_NEEDED
Intrinsics and/or Assembly

One of the promises of NaCl is access to the performance that comes from hand-tuning while not sacrificing portability or safety.

How do we get to, e.g., AES instructions, etc.?
How do we optimize for cache configuration, etc.?
Other future work

Clang

Other languages that could target bitcode
  .NET/Mono, ...

JIT support

Performance
  feedback directed optimization, ...

Bitcode size

Translation time
Want to Learn More?

http://www.chromium.org/nativeclient
(Follow Portable Native Client link)
http://code.google.com/p/nativeclient