LLVM MCJIT and debugging JIT-ted code

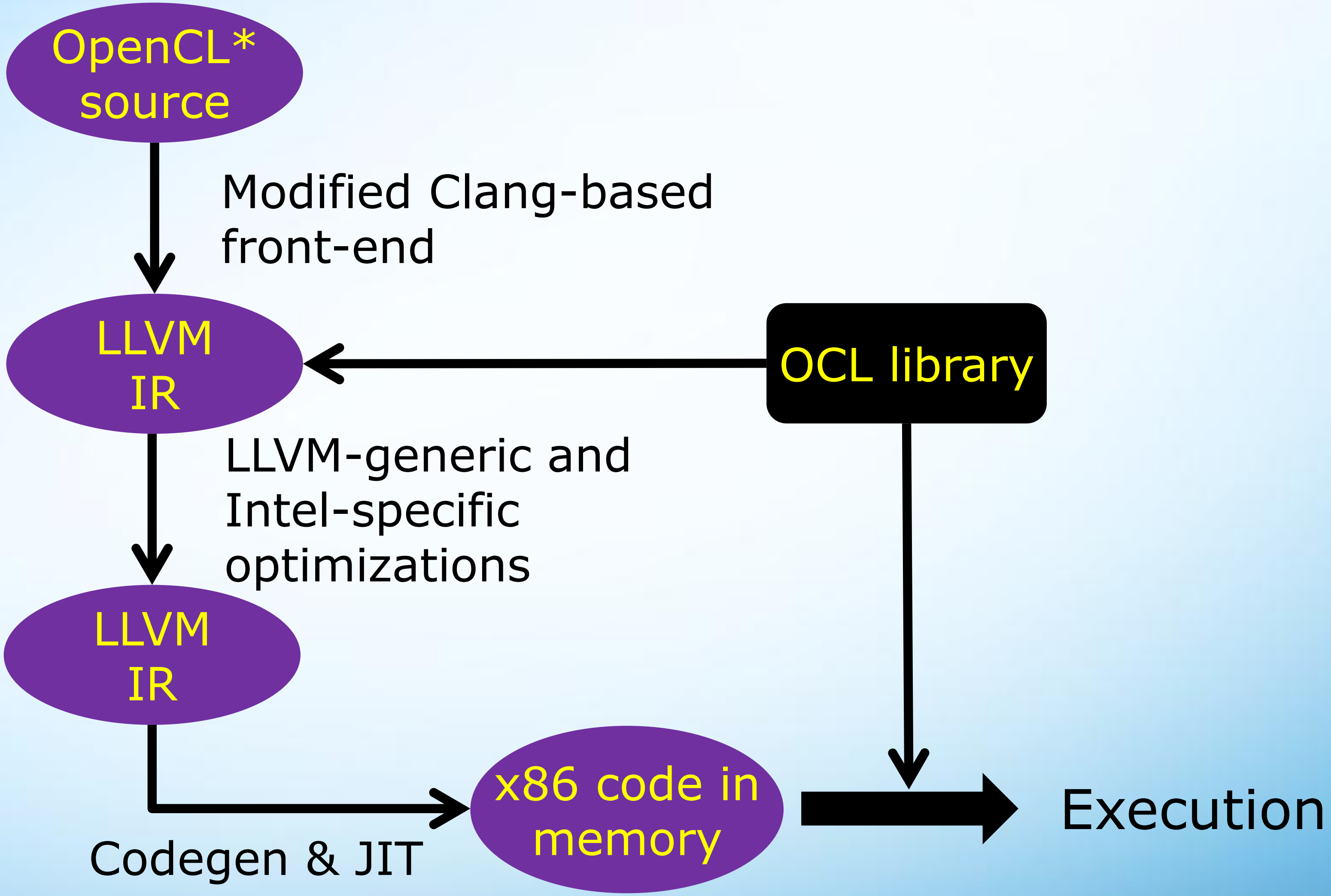
Eli Bendersky
[*With: Andy Kaylor, Matt Kopec, Daniel Malea, Ashok Thirumurthi*]

Intel® Developer Products Division
April 2012

Intel OpenCL* Team

- Responsible for the Intel SDK for OpenCL* applications.
- Develops LLVM-based OpenCL* compilers and tools.
- Enables future Intel® Architectures based on LLVM.
- The group is centered in Haifa, Israel:
  - With teams in California and Russia.
  - The MCJIT work is done in collaboration with a team in Waterloo, Canada.

Our OpenCL* compiler – high-level flow



OpenCL* source

Modified Clang-based front-end

LLVM IR

OCL library

LLVM-generic and Intel-specific optimizations

LLVM IR

Codegen & JIT

x86 code in memory

Execution

Our motivation – debugging JIT-ted code

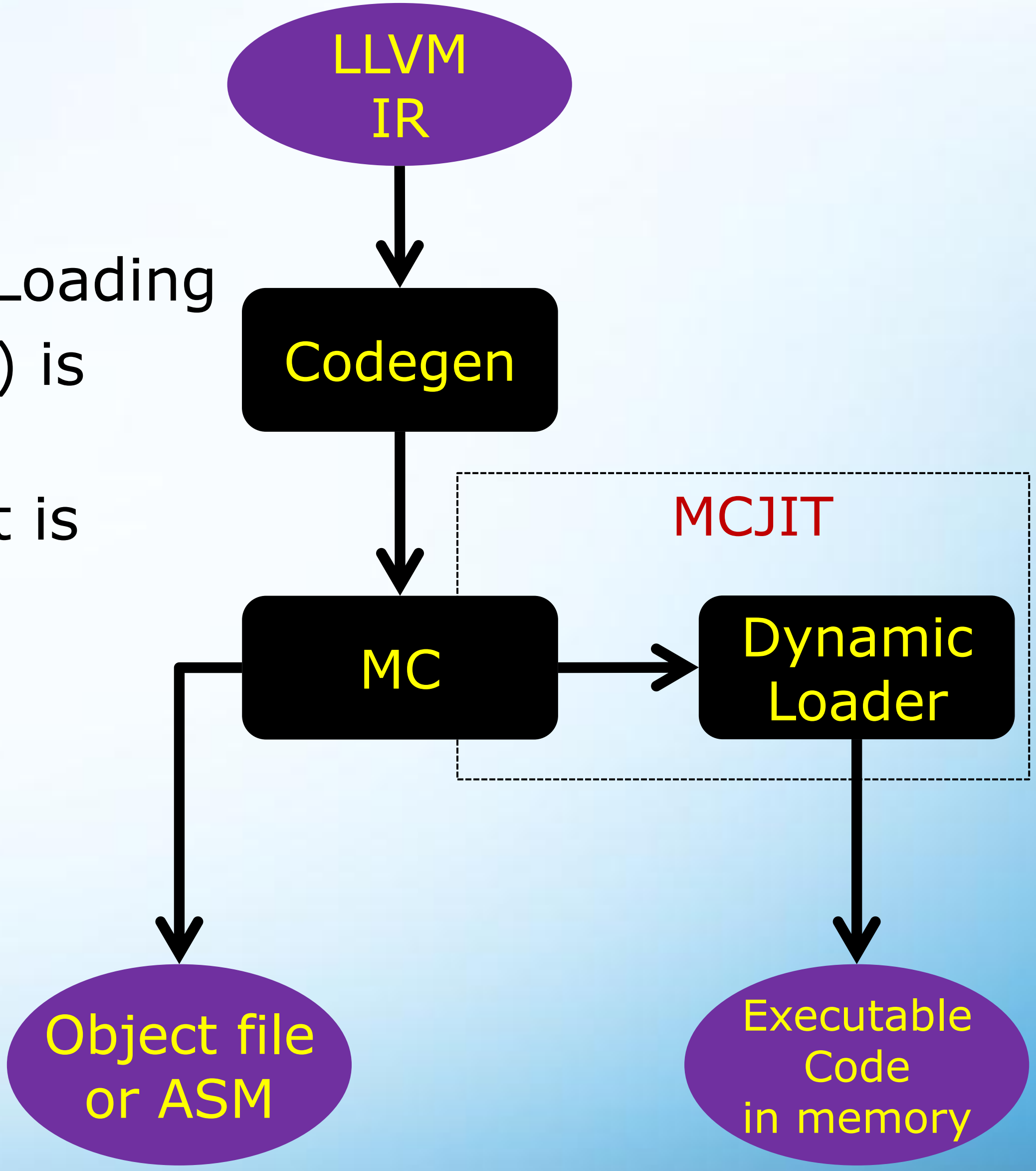
- A JIT interface has been added to GDB in version 7.0:
  - Runtime registration of JIT-ted objects for debugging.
- In LLVM, the JIT only emits frame information.
  - Function names in JIT-ted code when examining core dumps.
- The (old) JIT path does not support emitting full DWARF information.
  - Adding this is a lot of effort.

Debugging JIT-ted code – a solution

- Why not reuse the existing DWARF emitter in MC?
  - MCJIT !

MCJIT

- MCJIT is:
  MC + Runtime Dynamic Loading
- No separate emitter (JIT) is needed.
- The MC-generated object is dynamically loaded into memory and executed.



Motivation for MCJIT

- Avoid duplicate encoding paths (JIT & MC).
- Need to handle inline assembly.
- Debugging.

MCJIT – how it works

- Implements the `ExecutionEngine` interface.
- Accepts a custom memory manager (`JITMemoryManager` interface), target information, and a module to JIT.
- Runs passes from `TargetMachine::addPassesToEmitMC`:
  - `addPassesToGenerateCode` – common CodeGen passes.
  - `createMCObjectStreamer` – emit object code.
  - This creates an object file in a memory buffer.
- Uses the runtime dynamic loader (`RuntimeDyld`) to load the object and perform relocations necessary for execution.

MCJIT – Runtime Dynamic Loader

- The problem:
  - MC emits an object file (.o)
  - Object files are not executable – need to be *linked* and *loaded:*
    - Linked: resolve relocations between call sites and symbols.
    - Loaded: resolve absolute addresses, allocate BSS sections and COMMON symbols, resolve calls to other shared objects, etc.

MCJIT – Runtime Dynamic Loader

- Solution: Runtime Dynamic Loader:
  - A “linking loader”.
  - Do just enough linking and loading to make the object file executable in JIT.
- `RuntimeDyld` – a generic object:
  - Loads appropriate implementation (`RuntimeDyldImpl` interface), depending on object file identification (in the `loadObject` method).
  - For MachO – `RuntimeDyldMachO`
  - For ELF - `RuntimeDyldELF`

Runtime dynamic loading of ELF objects

- Sections that require allocation (SHF_ALLOC) are allocated using the memory manager.
- The absolute addresses of symbols are recorded:
  - At this point we know the “load address” of the sections symbols belong to, so absolute addresses are available.
- COMMON symbols are collected and allocated in a single chunk of data.
- Relocations are fixed up accordingly.

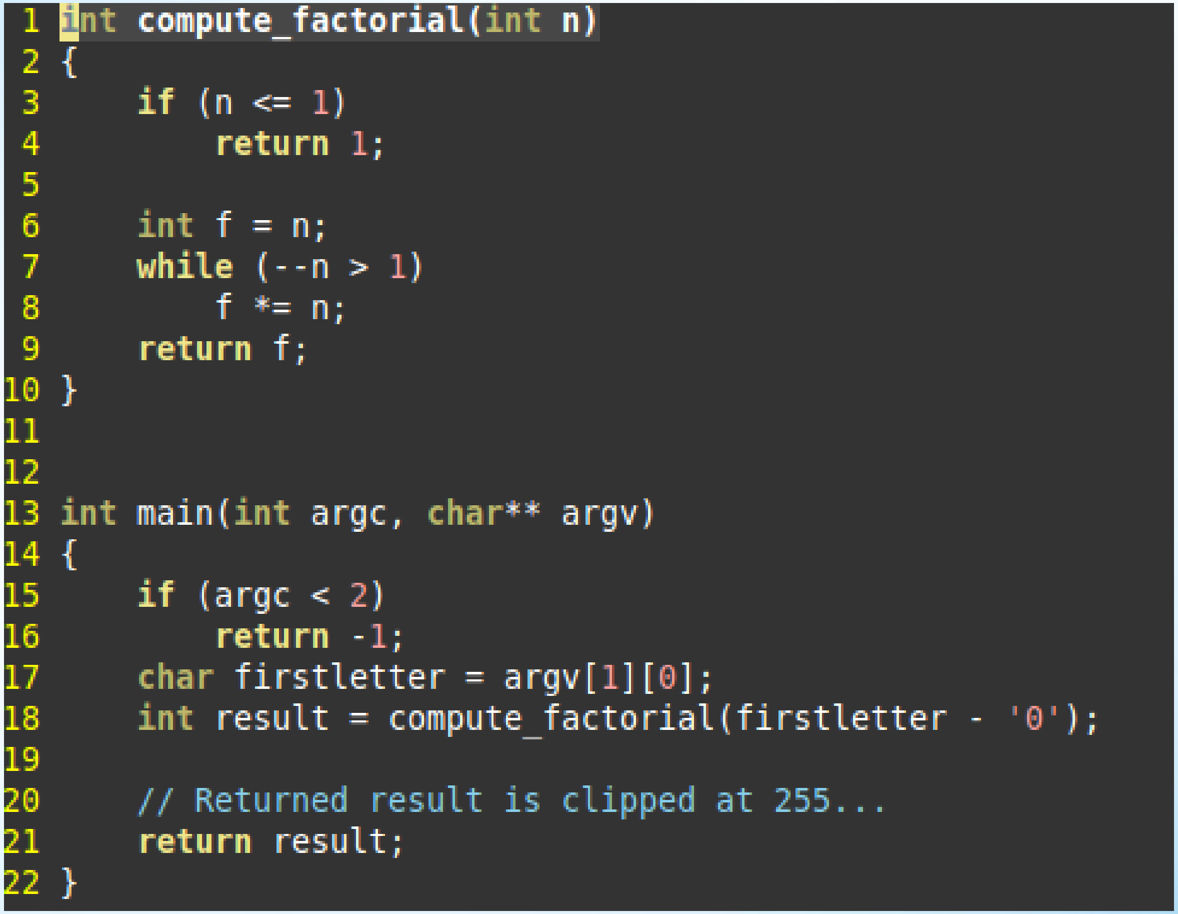
Debugging with MCJIT

- `JITRegistrar` – a singleton object responsible for registering JIT-ted objects with GDB.
  - Implements the GDB JIT interface.
- `RuntimeDyldImpl`, after performing relocations:
  - Calls `jitdebugging::registerObjectWithDebugger`

    → `JITRegistrar::registerObject`

    → `NotifyDebugger`
    - Fills the required GDB JIT interface data structures
    - Calls `__jit_debug_register_code`
- GDB just needs a pointer to the memory buffer holding the ELF image, and its size.

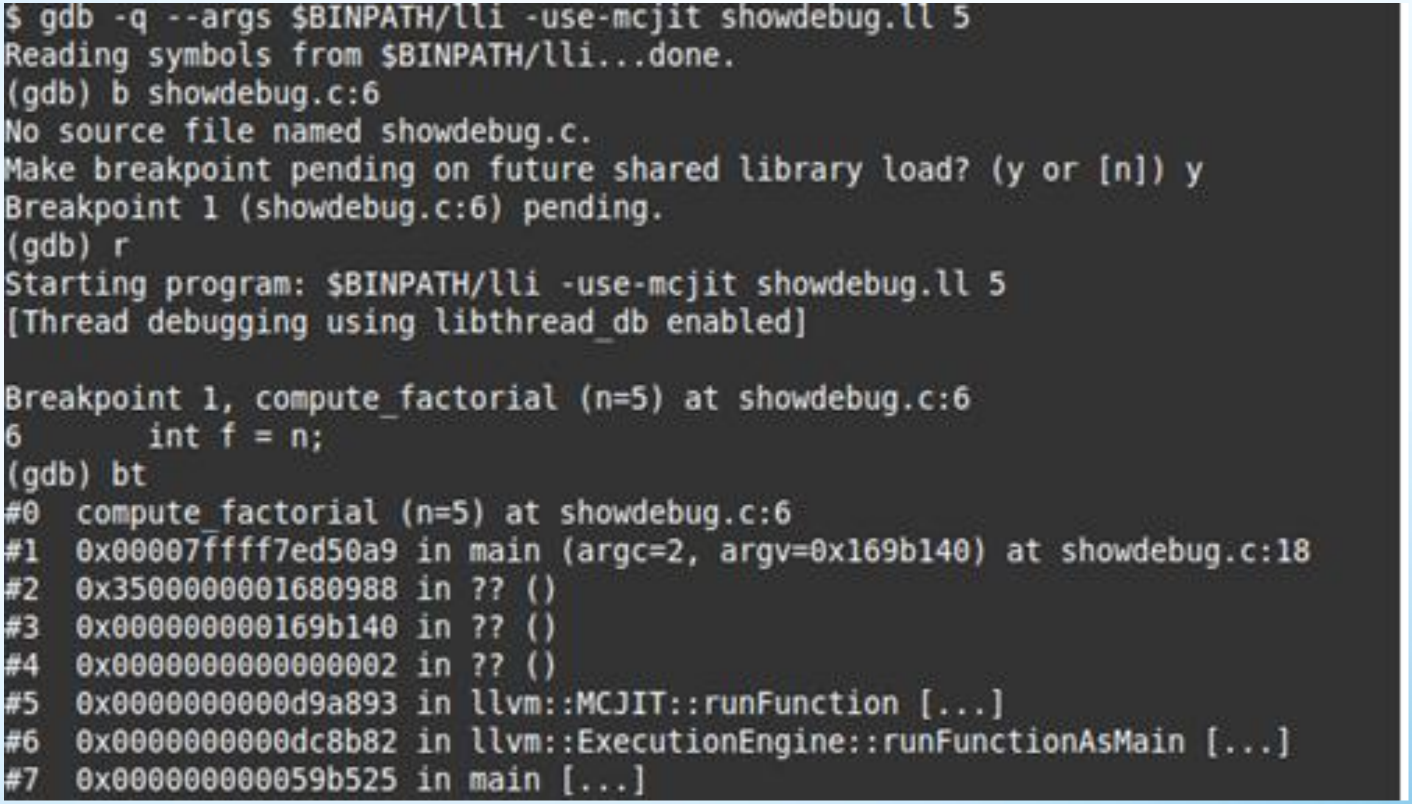
Demo – Debugging JIT-ted code



```c
int compute_factorial(int n)
{
    if (n <= 1)
        return 1;

    int f = n;
    while (--n > 1)
        f *= n;
    return f;
}


int main(int argc, char** argv)
{
    if (argc < 2)
        return -1;
    char firstletter = argv[1][0];
    int result = compute_factorial(firstletter - '0');

    // Returned result is clipped at 255...
    return result;
}
```

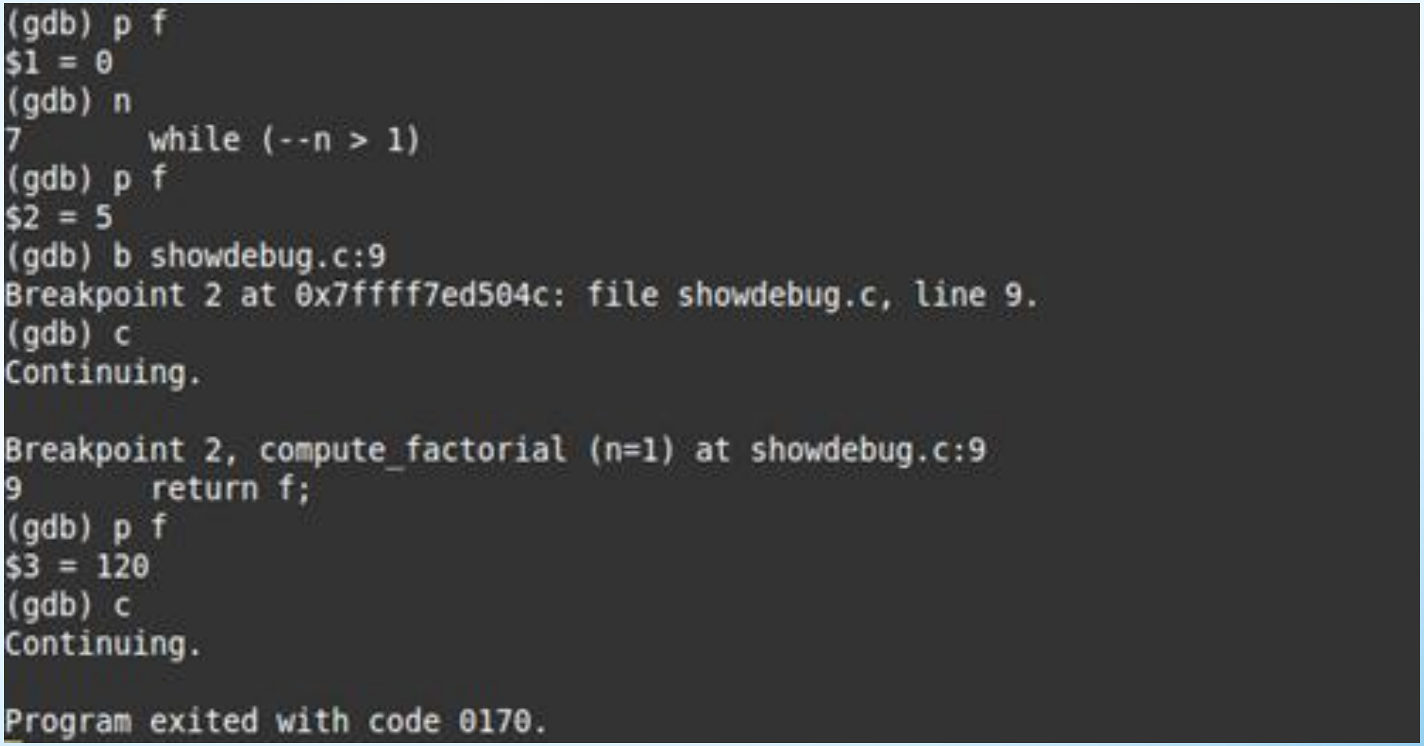
```
$BINPATH/clang -cc1 -g -O0 -emit-llvm showdebug.c
```

Demo – Debugging JIT-ted code



```
$ gdb -q --args $BINPATH/lli -use-mcjit showdebug.ll 5
Reading symbols from $BINPATH/lli...done.
(gdb) b showdebug.c:6
No source file named showdebug.c.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (showdebug.c:6) pending.
(gdb) r
Starting program: $BINPATH/lli -use-mcjit showdebug.ll 5
[Thread debugging using libthread_db enabled]

Breakpoint 1, compute_factorial (n=5) at showdebug.c:6
6         int f = n;
(gdb) bt
#0  compute_factorial (n=5) at showdebug.c:6
#1  0x00007ffff7ed50a9 in main (argc=2, argv=0x169b140) at showdebug.c:18
#2  0x3500000001680988 in ?? ()
#3  0x000000000169b140 in ?? ()
#4  0x0000000000000002 in ?? ()
#5  0x0000000000d9a893 in llvm::MCJIT::runFunction [...]
#6  0x0000000000dc8b82 in llvm::ExecutionEngine::runFunctionAsMain [...]
#7  0x000000000059b525 in main [...]
```

Demo – Debugging JIT-ted code



```
(gdb) p f
$1 = 0
(gdb) n
7         while (--n > 1)
(gdb) p f
$2 = 5
(gdb) b showdebug.c:9
Breakpoint 2 at 0x7ffff7ed504c: file showdebug.c, line 9.
(gdb) c
Continuing.

Breakpoint 2, compute_factorial (n=1) at showdebug.c:9
9         return f;
(gdb) p f
$3 = 120
(gdb) c
Continuing.

Program exited with code 0170.
```

Remaining challenges

- Efficiency:
  - Redundant copying of buffers (encumbered by the need to allocate both executable and non-executable buffers).
  - Compiling too much (old JIT only compiles reachable code).
- Windows* OS (ELF & triple):
  - Idea: load ELF on Windows* as well.
  - Challenge: the Triple enforces COFF generation on Windows*.
- Multiple modules:
  - Currently only a single module can be loaded into MCJIT.
  - MCJIT-ting multiple modules is challenging – linkage required.

Status of patches

Help is welcome

Optimization Notice

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

* “OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos

http://intel.com/software/products