Porting GCC to the AMD64 architecture

Jan Hubička

hubicka@suse.cz

SuSE ČR s. r. o.

AMD64 architecture

- New architecture used by 8th generation CPUs by AMD
- AMD Opteron
- AMD Athlon64

AMD64 architecture

- New architecture used by 8th generation CPUs by AMD
- AMD Opteron
  - Available since April
  - Server CPU
  - SMP capable
- AMD Athlon64

AMD64 architecture

- New architecture used by 8th generation CPUs by AMD
- AMD Opteron
  - Available since April
  - Server CPU
  - SMP capable
- AMD Athlon64
  - To arrive in September
  - Desktop CPU

AMD64 instruction set

- 64-bit extension of the existing IA-32 ISA.
- Single byte encodings of `inc` and `dec` removed.
- Four additional bits for instruction encoding via `rex` prefix

AMD64 instruction set

- 64-bit extension of the existing IA-32 ISA.
- Single byte encodings of `inc` and `dec` removed.
- Four additional bits for instruction encoding via `rex` prefix

  Bit 1 overwrite operand size to 64-bit

```
        89  c3  movl  %eax, %ebx
48      89  c3  movq  %rax, %rbx
41      89  c3  movl  %eax, %r10
44      89  c3  movl  %r8, %ebx
```

AMD64 instruction set

- 64-bit extension of the existing IA-32 ISA.
- Single byte encodings of `inc` and `dec` removed.
- Four additional bits for instruction encoding via `rex` prefix

  Bit 1 overwrite operand size to 64-bit

```
     89  c3  movl  %eax, %ebx
48   89  c3  movq  %rax, %rbx
41   89  c3  movl  %eax, %r10
44   89  c3  movl  %r8, %ebx
```

AMD64 instruction set

- 64-bit extension of the existing IA-32 ISA.
- Single byte encodings of `inc` and `dec` removed.
- Four additional bits for instruction encoding via `rex` prefix

  Bits 2–4 double amount of integer and SSE registers

```
     89  c3  movl  %eax, %ebx
48   89  c3  movq  %rax, %rbx
41   89  c3  movl  %eax, %r10
44   89  c3  movl  %r8, %ebx
```

AMD64 instruction set

- 64-bit extension of the existing IA-32 ISA.
- Single byte encodings of `inc` and `dec` removed.
- Four additional bits for instruction encoding via `rex` prefix
- IP relative addressing is cheaper than direct addressing
- Immediates remains 32-bit sign extended

```
89 04 25 34 12 00 00 mov %eax,0x1234
89 05 34 12 00 00    mov %eax,0x1234(%rip)
```

AMD64 instruction set

- 64-bit extension of the existing IA-32 ISA.
- Single byte encodings of `inc` and `dec` removed.
- Four additional bits for instruction encoding via `rex` prefix
- IP relative addressing is cheaper than direct addressing
- Immediates remains 32-bit sign extended
- 32-bit operations zero extend
- `movabs` instruction to load 64-bit immediates available

GCC porting effort

- AMD disclosed specification and simulator early
  - Open development of GCC and Binutils
  - Discussed in the public mailing lists
    http://www.x86-64.org
  - Implemented mostly by small team in SuSE
  - The first available compiler for the platform

GCC porting effort

- AMD disclosed specification and simulator early

August 10, 2000	Specs
September, 2000	First GNU tools
October 6, 2000	Simulator
January 16, 2001	Emulator, working Linux kernel
January 8, 2002	AMD64 in GCC mainline
Februrary 26, 2002	First Hardware, working userland, X11
August 15, 2002	GCC 3.2
April 22, 2003	AMD Opteron, SuSE GNU/Linux

GCC porting effort

- AMD disclosed specification and simulator early
- Porting was easy
  - Hit just very few limitations of generic GCC code
  - Majority of problems caused Binutils port (lack of experience and crazy design)
  - Enough time to concentrate on code quality

GCC porting effort

- AMD disclosed specification and simulator early
- Porting was easy
- System V Application Binary Interface; AMD64 Processor Supplement (PSABI)
  - First compiler port has chance to design ABI
  - Draft versions released and discussed in public
  - GCC used to test decisions

PSABI

- Based on IA-32 PSABI
- Important changes in:
  - Natural alignment for all types
  - `sizeof(long double)`=16 (not 12).
    - 48-bits of padding for fast loads/stores
  - Register argument passing conventions
  - Multiple code models
  - PIC take advantage of IP relative addressing
  - Stack unwinding done using DWARF2
  - Red zone to reduce amount of stack frame allocations

Argument passing

- 6 Integer and 6 SSE registers used to pass arguments
- 2 integer, 2 SSE and 2 x87 registers used to return values
- Small ($\leq$ 16 bytes) aggregates are passed in registers
- Arguments are passed in registers only when they fit as a whole
- Otherwise passed by value (unlike PPC PSABI)
- Arguments are not promoted to 64-bits (breaks some programs, like GNOME)

Variadic argument

- `va_list` type definition

```
typedef struct {
   unsigned int gp_offset;
   unsigned int fp_offset;
   void *overflow_arg_area;
   void *reg_save_area;
} va_list[1];
```

Variadic argument

- `va_list` type definition

```
typedef struct {
   unsigned int gp_offset;
   unsigned int fp_offset;
   void *overflow_arg_area;
   void *reg_save_area;
} va_list[1];
```

- Register `al` used to specify amount of SSE operand registers when calling variadic or prototypeless function
  - Jump-table used in the prologues
  - Breaks non-conforming programs calling variadic functions with non-variadic prototype (strace)

Variadic argument

- `va_list` type definition

```
typedef struct {
   unsigned int gp_offset;
   unsigned int fp_offset;
   void *overflow_arg_area;
   void *reg_save_area;
} va_list[1];
```

- Breaks non conforming code:

```
void t (va_list *);
void q (va_list a)
 {
   t(&a);
 }
```

Code models

- Limiting code and data size save resources
  - **small model** code+data in the low 31-bits
  - **medium model** code limited, data unlimited
    3.28% slowdown, 14% code size growth
  - **large mode** unlimited, not implemented

Code models

- Limiting code and data size save resources
  - **small model** code+data in the low 31-bits
  - **medium model** code limited, data unlimited 3.28% slowdown, 14% code size growth
  - **large mode** unlimited, not implemented
- Position Independent Code
  - GOT, PLT and static data accessed by IP relative addressing
  - Limited to 31bits
  - Large PIC libraries would need new relocations
  - 8% performance penalty on AMD64, 20% on IA-32
  - 5% growth AMD64, 0.8% on IA-32

DWARF2 Stack Unwinding

- DWARF2 already used for exception handling
- First GCC target to use DWARF2 everywhere
  - Pros:
  - Cons:

DWARF2 Stack Unwinding

- DWARF2 already used for exception handling
- First GCC target to use DWARF2 everywhere
  - Pros:
    - Give maximal freedom of stack frame optimizations
    - Standard and used for EH in GCC
    - Asynchronous unwind information generated by default is useful for garbage collection and debugging
  - Cons:

DWARF2 Stack Unwinding

- DWARF2 already used for exception handling
- First GCC target to use DWARF2 everywhere
  - Pros:
  - Cons:
    - Complex implementation
      - PSABI specify unwinding API compatible with IA-64
    - Large unwind tables (7% of the code size)
    - Unwinding is relatively slow
    - Need to manually annotate assembly functions
      - GAS extension urgently needed

DWARF2 Stack Unwinding

- DWARF2 already used for exception handling
- First GCC target to use DWARF2 everywhere
  - Pros:
  - Cons:
- New support for the unwinding in GDB
- may be considered for other platforms too.
- s390 already adopted the scheme
- We should consider the shift for 386 to enable `-fomit-frame-pointer` by default (3.26% speedup, 0.95% growth on IA-32)

Floating point math

- i386 use x87 instruction set to implement floating point arithmetics
- SSE Instruction set

Floating point math

- i386 use x87 instruction set to implement floating point arithmetics
  - All temporary registers are 80-bit values
    - Produced code is not IEEE compliant
    - Some algorithms fail to converge
    - Results depend on compiler optimization
    - Some algorithms enjoy the extra precision for free
  - Registers do have stack organization wasting performance
- SSE Instruction set

Floating point math

- i386 use x87 instruction set to implement floating point arithmetics
- SSE Instruction set
  - Originally designed for single precision vector code
  - Support for scalar operations
  - Later extended for double precision support
  - Generally useful IEEE compliant fbating point unit with few oddities
  - `neg` and `abs` done by vector logic operations
  - 128-bit registers
  - Fewer instructions, bigger code (1.6–7.3% growth)

Floating point math

- i386 use x87 instruction set to implement floating point arithmetics
- SSE Instruction set
- GCC use SSE for single and double precision, x87 for extended precision
- Huge performance impact for both 64-bit and 32-bit code
- Can we adopt the scheme by default on IA-32?
  - Some programs rely on the 80-bit temporaries
  - FLT_EVAL_METHOD is not invariant for all units

The AMD Opteron

- Translate AMD64 instruction into micro instruction
- Register renaming, on-chip scheduling
- 3 integer, 3 address generation symmetric pipes
- Floating point load/store, adder and multiplier pipes
- Speculative loads and stores with on-chip memory controller allowing operations to be canceled early

Optimizing for the Opteron

- Many optimizations done in hardware
- Almost insensitive for integer instruction code choice
- Important optimizations
  - Avoid SSE reformatting penalties
  - Use right SSE move instructions
  - Optimize prologues and epilogues to avoid function call bottleneck
  - Avoid use of push and pop instructions for argument passing in hot code
  - Local scheduling is still effective, global scheduling may be important win

Optimizing for the Opteron

- Many optimizations done in hardware
- Almost insensitive for integer instruction code choice
- Important optimizations
- 10% speedup compared to 386 tuning
- 1.13% speedup compared to Pentium II tuning

Optimizations

Effect on SPECint2000

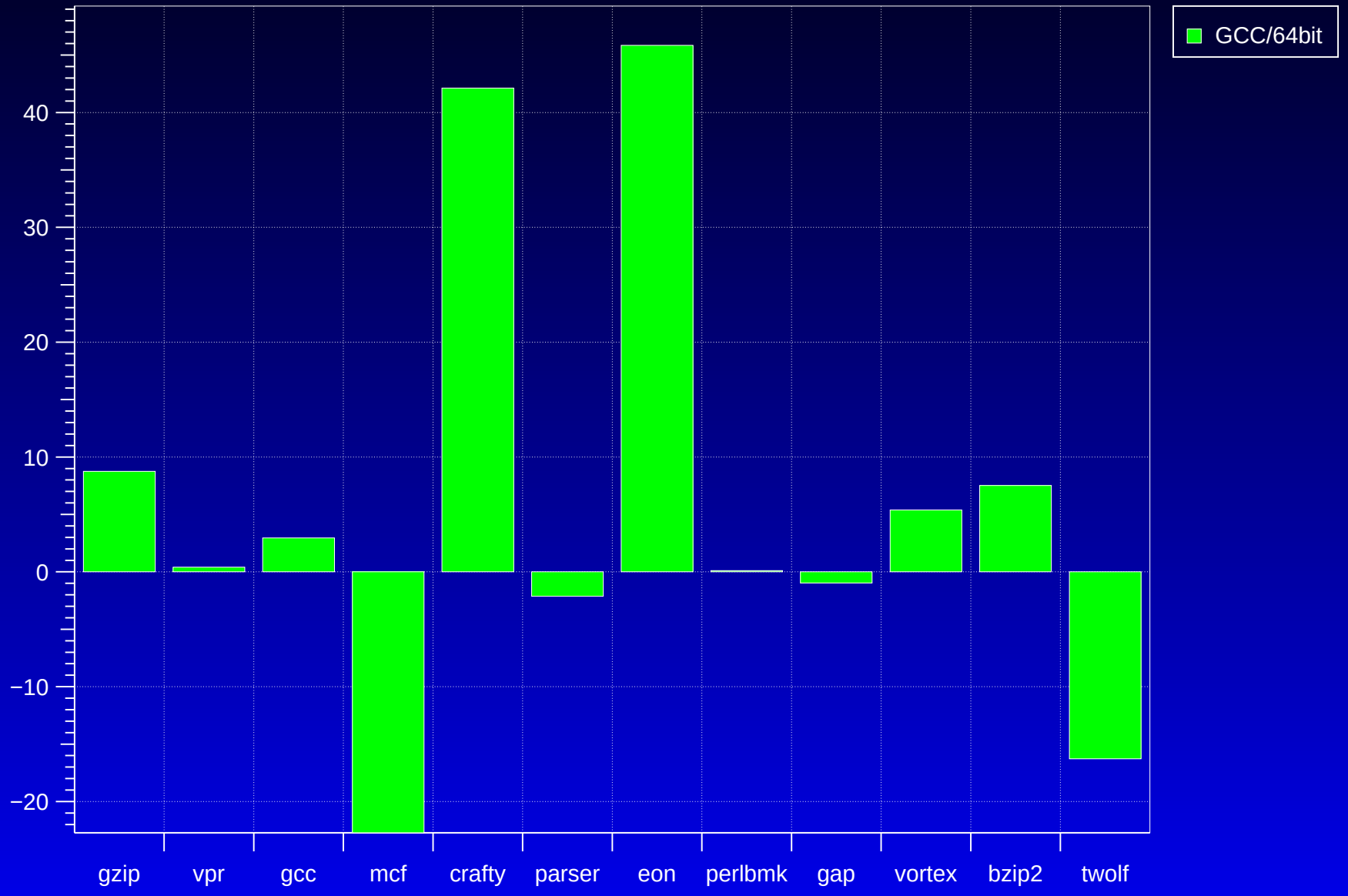
Optimization	64-bit	32-bit	Alpha
overall effect	106.9%	89.12%	115.07%
profile feedback	9.49%	7.37%	6.16%
loop unrolling	3.12%	1.25%	1.30%
basic block reorder	2.18%	1.26%	2.64%
omitting frame pointer	2.10%	3.26%	2.64%
function inlining	1.85%	2.17%	6.84%
tracer	1.60%	1.78%	−2.59%
scheduling	1.47%	1.52%	8.08%

Optimizations

Effect on SPECfp2000 (C, f77 only)

Optimization	64-bit	32-bit	Alpha
overall effect	149.9%	98.56%	115.07%
use of SSE	13.80%	10.14%	
profile feedback	3.39%	1.66%	3.14%
scheduling	1.41%	−0.72%	21.69%
tracer	0.15%	−0.37%	2.36%

SPECint2000 32-bit to 64-bit



LP64 problems

- 64-bit pointers increase data structures
- Combining integer and pointer arithmetics introduces sign extensions

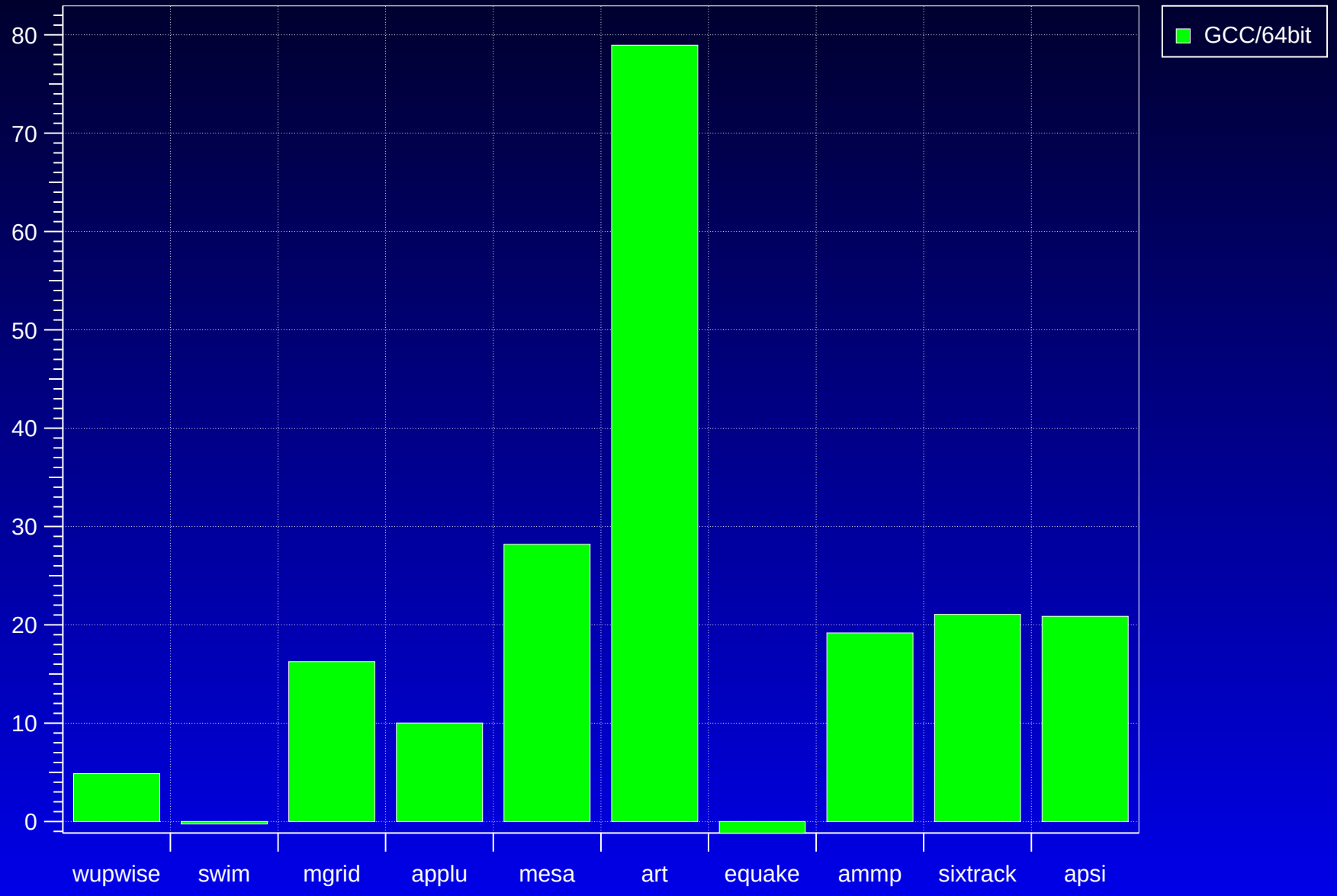
LP64 problems

- 64-bit pointers increase data structures
  - Memory bound pointer chrunking slows down
  - Larger programs load more slowly
  - Higher memory requirements
  - The faster CPU, the worse relative performance
  - Can be solved by adding “tiny model” with 32-bit pointers in 64-bit mode.
    - Lose benefits of 64-bit system
    - Require kernel emulation layer, glibc support,. . .
    - Perhaps just a benchmark trick
- Combining integer and pointer arithmetics introduces sign extensions

LP64 problems

- 64-bit pointers increase data structures
- Combining integer and pointer arithmetics introduces sign extensions
  - Undefined overflows allow cheap conversion of 32-bit arithmetics to 64-bit
  - Not applicable. AMD64 has cheaper 32-bit operations than 64-bit operations
  - Can be converted to zero extensions
  - Zero extensions are often for free
  - Still limits use of addressing modes

SPECfp2000 32-bit to 64-bit



Comparison to 32-bit system

- AMD should provide first cheap 64-bit box
- Performance

test	speedup
SPECint2000	3.4%
SPECfp2000 (C/f77)	19.3%
bootup time	−0.9%
KDE startup from disk	18.1%
KDE startup from cache	14.6%
compilation	12.9%
`./configure`	−4.3%

Comparison to 32-bit system

- AMD should provide first cheap 64-bit box
- Performance
- Memory consumption

test	growth
konqueror	28%
gimp	15%
mozilla	22%

Comparison to 32-bit system

- AMD should provide first cheap 64-bit box
- Performance
- Memory consumption
- Binary file sizes

section	growth
code section	−5%
data sections	37%
unwind info	1414%
total	13%

Future work

- Proper implementation of 128 bit integer arithmetics (partial hardware support is available)
- GAS support for Dwarf2 unwinding
- Implementation of `__float128`, true 128bit floating point type in software emulation
- More optimizations

Future work

- Proper implementation of 128 bit integer arithmetics (partial hardware support is available)
- GAS support for Dwarf2 unwinding
- Implementation of `__float128`, true 128bit floating point type in software emulation
- More optimizations
- More fun :)

Credits

Geert Bosch designed stack unwinding and exception handling ABI. Richard Brunner, Alex Dreyzen and Evandro Menezes provided a lot of help in understanding the AMD Opteron hardware. Zdeněk Dvořák implemented the new loop unrolling pass, improved DWARF2 support and did number of improvements to profile based optimizations framework. Andrew Haley finished the gcj (Java compiler) port started by Bo Thorsen. Richard Henderson reviewed majority of the GCC changes. Jan Hubička implemented the first versions of GCC and Binutils ports, co-edited ABI document, realized the AMD Opteron specific optimizations and some generic ones (unit at a time mode, profile feedback optimizations framework, tracer). Andreas Jaeger ported glibc, provided SPEC2000 testing framework, co-edited ABI document and fixed number of GCC and Binutils bugs. Jakub Jelínek designed and implemented the thread local storage ABI. Michal Ludvig and Jiří Šmíd realized the GDB port. Michael Matz worked on the new register allocator and fixed plenty of GCC bugs. Mark Mitchell edited the ABI document and set up WWW and CVS of the project. Andreas Schwab and Bo Thorsen fixed number of problems in the linker and assembler. Josef Zlomek redesigned the basic block reordering pass and fixed number of bugs in GCC.