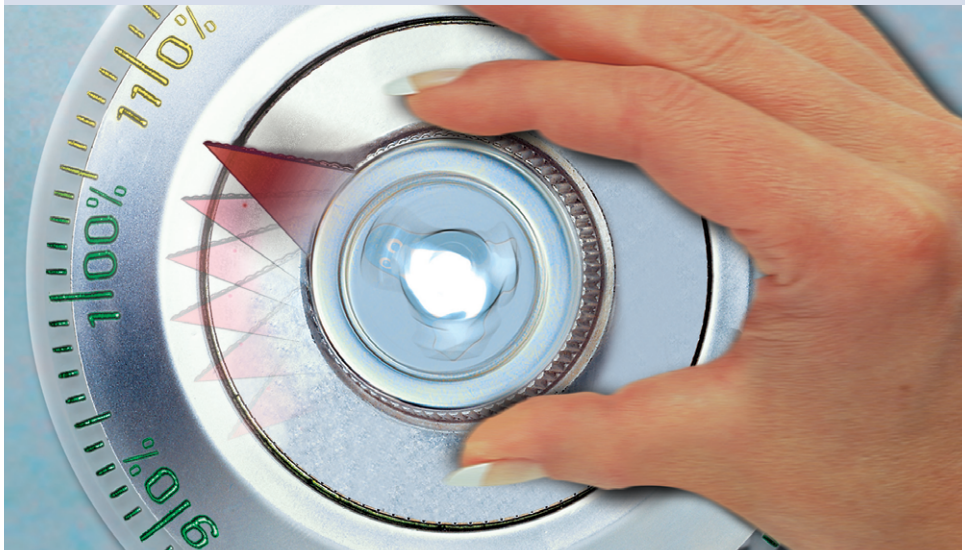
Linux-Kernel mit dem Intel-Compiler übersetzen

Aufgedreht: Kernel-Tuning

Der Intel-C-Compiler erzeugt meist spürbar schnelleren Code als der GCC. Leider scheiterte bisher jeder Versuch, den Linux-Kernel 2.6 durch das Tool zu schleusen. Dieser Exklusivbericht beweist, wie es trotzdem gelingt, und belegt die entstehende Performancesteigerung. Ingo A. Kubbilun



Mit dem GCC 3.3.3 ist dem Intel-C/C++-Compiler ICC ein würdiger Gegner erwachsen, der ebenfalls die Merkmale moderner Prozessoren einschließlich des Pentium 4 und artverwandter AMD-CPUs unterstützt. Trotzdem profitiert die Geschwindigkeit vieler Programme und Bibliotheken, die mit den hoch optimierenden Methoden des ICC übersetzt wurden, gegenüber den GCC-Kompilaten (siehe Linux-Magazin-Artikel **[1]**, **[2]**).

Da liegt der Wunsch nahe, auch den Linux-Kern mit dem Intel-Compiler zu übersetzen. Zwar wird wohl der zu erwartende Geschwindigkeitsschub im Vergleich zu dem von Anwendungen oder Bibliotheken kleiner ausfallen – schließlich fungiert der Kernel hauptsächlich als Vermittler zwischen Hardware und Benutzerschicht und ist daher stark E/A-lastig. Doch wird dieser Artikel belegen, dass eine messbare Performancesteigerung ohne weiteres möglich ist.

Anders als der GCC fallen für den Intel-Compiler im kommerziellen Anwendungsfall rund 400 US-Dollar Lizenzkosten an. Intel hat aber ein Herz für Linuxer, die den Übersetzer privat nutzen, und vergibt eine kostenfreie „Free non-commercial unsupported version“. Wenn man als Studienobjekte den Kern 2.6.5 und den ICC 8.0.055 **[3]** heranzieht, wird schnell klar, dass – anders als Intel in **[4]** behauptet – das Übersetzen des Kernels unter Einbeziehung von ein paar Patches nicht gelingt, zumindest beim 2.6 nicht (siehe Abbildung 1).

Schwere Kost

Die Bekundungen von Intel, der ICC 8.0 sei Quellcode- und binärkompatibel zum GCC 3.2, treffen für das Übersetzen des Linux-Kerns nur sehr bedingt zu. In der Theorie verläuft ein Übersetzerwechsel beim Kernel einfach: Man tauscht im Top-Level-Makefile den »gcc« durch den neuen Compiler aus. Doch der Linux-Quellcode ist für jeden Compiler eine schwere Kost, bei der sich herausstellt, dass beide Übersetzer eben nicht 100-prozentig kompatibel sind.

Der Begriff Übersetzerkompatibilität bedarf einiger Erläuterungen. Generell zeigen verschiedene Übersetzer bei ihren Kommandozeilenoptionen, beim erzeugten Code und der Emittierung von Daten, bei den eingebauten Funktionen (den so genannten Intrinsics), den Assembler-Sequenzen und bei allen Dingen, die über den Ansi/ISO-C-Standard hinausgehen, unterschiedliches Verhalten. Das legt nahe, dass die Substitution des GCC diverse Modifikationen an den Kern-Quellcodes notwendig macht.

Derzeit (Intel-Compiler 8.0, Patchlevel 055) reichen jedoch ein paar solcher Änderungen in den Quellen (Kernel 2.6.5) nicht aus, um ihn mit dem ICC sauber übersetzen zu können. Das hier vorgestellte Patch (Download unter **[5]**) bedient sich darum zweier kleiner Helferprogramme, des »ccd« (C Compiler Delegate) und des »lkd« (Linker Delegate). Statt Intels C-Compiler beziehungsweise den Linker gleich direkt aufzurufen,

Linux* Kernel Build

Intel C++ Compiler 8.0 for Linux has greatly improved support for GNU C language extensions. As a demonstration of compatibility with gcc, Intel C++ Compiler 8.0 for Linux has successfully built and run Linux kernels on IA-32 and Itanium processors using a limited number of temporary source patches. The temporary nature of these patches will be addressed in the future either by adding extensions to the Intel C++ Compiler or by working with the Linux community to replace or reduce usage of less frequently used, non-standard language features in the kernel sources. For additional details on the Linux kernel build, see support.intel.com/support/performancetools/c/linux/sb/CS-007713.htm

Note: that while we are interested in your feedback on Intel Compilers, any questions directly related to the kernel implementation need to be supported by the Linux kernel-build community. To report issues concerning the use of Intel Compilers to build the Linux kernel, please use Intel Premier Support, which is included with every purchased compiler.

Abbildung 1: Anders als Intel hier in [4] behauptet, gelingt dem ICC das Übersetzen des Kernels mit ein paar Patches nicht.

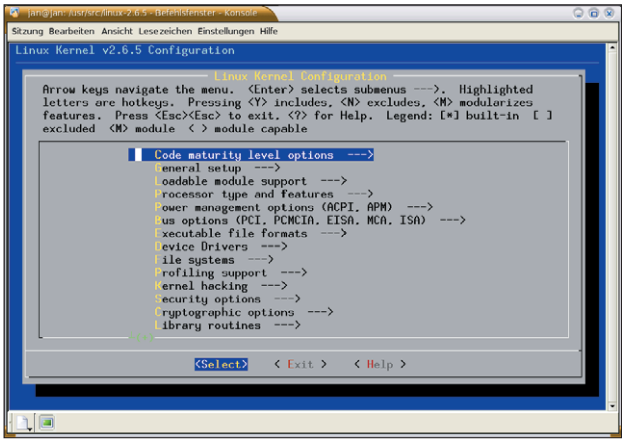


Abbildung 2: Bevor der Intel-Compiler ins Spiel kommt, konfiguriert man die Kernelquellen entsprechend der vorliegenden Hardware.

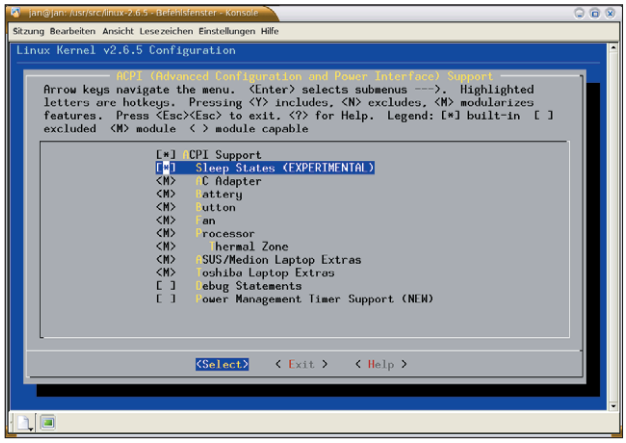


Abbildung 3: Besonders die als experimentell gekennzeichneten Kernelmodule müssen mit dem GCC klaglos laufen.

schalten sich die beiden Tools als Vermittler dazwischen.

Das Duo vollführt kunstvoll diverse Nachbearbeitungsschritte und begegnet so den Übersetzer-Inkompatibilitäten. C Compiler Delegate übersetzt den C-Quellcode mit dem Intel-Compiler zunächst in Assembler-Code und führt anschließend die im Folgenden beschriebenen Nachbearbeitungsschritte aus. Danach veranlasst »ccd« den GNU-Assembler dazu, aus dem modifizierten Assembler-Code Objektcodes zu erzeugen. Linker Delegate arbeitet analog, indem es den GNU-Linker fernsteuert.

Die so erzeugten Linux-Kerne laufen sehr stabil auf Ein- und Zwei-Prozessor-Maschinen mit Pentium III und 4, was sowohl eine Beschäftigung aus technischem Interesse rechtfertigt, als auch wegen des Performancevorteils ein Potenzial für Hardware-Einsparung ist. Doch Vorsicht: Evolutionär muss man ICC-übersetzte Kernel derzeit als Entwicklerversionen einstufen, die Admins erst nach Absolvierung von Langzeittests auf Produktivsystemen einsetzen sollten.

Kernel übersetzen – die Praxis

Zielgerichtetes Vorgehen bei der Übersetzer-Migration umfasst drei Schritte: Zunächst konfiguriert man den Kern der vorliegenden Hardware entsprechend, übersetzt ihn mit dem GCC und prüft, ob er funktioniert. Besonders ist darauf zu achten, dass als experimentell gekennzeichneten Kernelmodule klaglos laufen (siehe Abbildung 2 und 3).

Der zweite Schritt unterzieht den Quellcode allgemeinen Modifikationen, sodass der Intel-Compiler einen sauberen Übersetzungslauf hinlegt. (Hinweise auf die Installation des ICC liefert der Kasten „ICC-Installation“.) Das Adjektiv sauber ist beim Intel-Compiler zu relativieren, denn er ist auf der üblichen Warnstufe »-Wall« wesentlich penibler als der GCC und überflutet die Konsole mit allerlei Warnungen über unterlassene Typumwandlungen sowie Zeiger-Arithmetik mit »void«- und Funktionspointern. Das Kernelpatch bedient sich daher des Schalters »-w«, um nur echte Übersetzungsfehler anzuzeigen.

Einige wenige Stellen des Kerns übersetzt der ICC leider zu falschem Objektcode. Sie aufspüren und beheben schließt die zweite Phase ab. Im dritten und letzten Schritt passieren Intel-spezifische Modifikationen, die zwei besondere Eigenschaften des Intel-Compilers nutzen: IPO (Inter Procedural Optimization) und PGO (Profile Guided Optimization). Insbesondere die PGO-Instrumentalisierung von Kernelcode bedarf eines hohen technischen Aufwands – der macht sich jedoch bezahlt. ▶

ICC-Installation

Die Installation von Intels C++ Compiler 8 für Linux erwartet mindestens 100 MByte freien Speicherplatz und eine Glibc-Version 2.2.5, 2.2.93 oder 2.3.2. Zuerst muss man auf der Intel-Homepage [3] ein kleines Formular ausfüllen und der Lizenz zustimmen. Die Lizenz erlaubt es nur, privaten Code zu übersetzten. Intel sendet eine E-Mail an die angegebene Adresse mit genauen Anweisungen.

Nach dem Download landet ein 64 MByte großes Tar.gz-Archiv auf der Platte. Die entpackten Dateien enthalten alle nötigen RPM-Files sowie das »install.sh«-Skript, das mit »source« aufzurufen ist. Das Installationsskript macht Gebrauch vom »rpm«-Kommando, weshalb es mit Root-Rechten laufen muss. Wer die nicht hat oder keine RPM-basierte Distribution, zerlegt laut »C++Release-Notes.htm« die RPM-Dateien mit »rpm2cpio«. Vor dem Start der Installation ist die Lizenzdatei aus der E-Mail in das Verzeichnis zu kopieren, das in der Variablen »INTEL_LICENSE_FILE« genannt ist. Diese Variable kann beliebig gesetzt sein, es empfiehlt sich aber die Defaultposition in »/opt/intel/licenses«.

Konfiguration

Nach der Installation des »icc« sind einige Einstellungen erforderlich. Liegt der Intel-Compiler in »/opt/intel/«, finden sich die Einstellungen für die Defaultoptionen in der Datei »/opt/intel/bin/icc.cfg«. Dieses File kann jeder an seine eigenen Vorstellungen anpassen.

Intel stellt Skripte zur Verfügung, die die Environment-Variablen »PATH« und »LD_LIBRARY_PATH« automatisch setzen. Das für seine Shell passende Skript sollte jeder Programmierer vor dem Kompilieren laden, etwa in die Bash:

```
. /opt/intel/bin/iccvars.sh
```

Auf Entwicklungsmaschinen sollte der Admin diese Befehle in die Startdateien der von ihm eingesetzten Shell aufnehmen, etwa in die »/etc/bashrc«.

Eine Eigenschaft moderner Prozessoren ist nach Studien des Autors dieses Artikels leider nicht anwendbar: Der Vektorisierer des Intel-Compilers zur Erzeugung von SIMD-Code (Single Instruction Multiple Data, [2]) scheitert beim Kernel, wie im Kasten „Warum SIMD nichts bringt“ nachzulesen ist. Auch ohne SIMD bleibt genug Raum, um hoch optimierten Kernelcode zu erzeugen.

Minimal invasiv

Eine goldene Regel des Patchens beherzigt auch das hier vorgestellte Set: Halte die Anzahl der Modifikationen so gering wie möglich! Das stellt sicher, dass die Kern-Modifikationen auch auf künftige Linux-Versionen mit geringem Aufwand übertragbar sind. Das Patch [5] renoviert diverse Make-Dateien des Kerns. Das erwähnte Werkzeug »ccd« hält Einzug in das Top-Level-Makefile und ersetzt den GNU-Compiler »gcc«. Die Installation des Patches erklärt die im Archiv enthaltene Datei »INSTALL«.

Einige Kommandozeilenoptionen des Intel-Compilers sind inkompatibel zu denen des GCC. Das gilt besonders für solche, die das Erzeugen von Rahmenzeigern (Frame Pointers) steuern oder eine Prozessorarchitektur festlegen. Auch gilt es, die richtigen Schalter auszuwählen, um SIMD-Instruktionen im Kern auszuschließen (in »arch/i386/Makefile«). Einige weitere marginale Änderungen führen ans primäre Ziel, das erfolgreiche Übersetzen des Kerns.

Die verbesserte Kompatibilität zum GNU-Compiler schlägt sich bereits in einigen Kleinigkeiten nieder. Bis einschließlich Version 7 erzeugte der Intel-Compiler leere Strukturen (zum Beispiel bei Spinlocks) zwar als leer, definierte deren Länge jedoch als eins. Die GNU-Compiler in den Versionen 2.91.*x* machten den gleichen Fehler.

Das Vorgaukeln der Übersetzerversion 2.91 könnte das Problem beheben, indem ein Element mit dem bezeichnenden Namen »gcc_is_buggy« in den eigentlich leeren Strukturen bedingt einkompiliert wird – besser müsste das Element »icc_is_buggy« heißen. Zum Glück hat Intel diesen Fehler in der achten Version des Übersetzers behoben, was auch den Workaround unnötig macht.

SIGSEGV

Ein bis hierhin modifizierter Kern ist leider noch immer Bitschrott und zudem funktionell eingeschränkt. Denn der Versuch, ein Kern-Modul mit gemeinen Symbolen (Common Symbols) zu laden, schlägt fehl. Das sind globale Variablen, die weder statisch noch initialisiert sind. Die Übersetzung aller Kern-Quellcodes erfolgt mit dem Schalter »-fno-common«, den auch der Intel-Compiler kennt. Leider ist die namensgleiche Option beim ICC anders implementiert. Das Werkzeug »ccd« ersetzt in einem Nachbearbeitungsschritt alle gemeinen Symbole durch gleichwertige Konstrukte und verlagert die Variablen in die Sektion ».bss«.

Daneben geht der Intel-Compiler recht eigenwillig mit manchen Attributen um, beispielsweise »__attribute__((used))«. So kennzeichnet er die exportierten Kern-Symbole als benutzt, obwohl sie der Kern-Code nicht referenziert. Das Attribut ist ein Hinweis an den Übersetzer, derartig gekennzeichnete Variablen und Funktionen beim Optimieren keinesfalls zu entfernen. Der IPO-Mechanismus des Intel-Compilers überlagert fälschlich die Wirkung dieses Attributs. Die Einführung von Dummy-Zugriffsfunktionen, die »ccd« vor der Assemblierung wieder entfernt, löst das Problem.

Das Laden des Moduls »ieee1394.ko« führt zu dem Ausnahmefehler »SIGSEGV«. Ein Blick in die ELF-Binärdatei des Moduls verrät, dass der Intel-Compiler falsche Relokationen für alternative Instruktionen erzeugt. Sofern man beispielsweise ein Modul für eine spezifische Prozessorarchitektur übersetzt und

Warum SIMD nichts bringt

Beide Übersetzer, GCC und ICC, beherrschen den Umgang mit SIMD-Instruktionen (Single Instruction Multiple Data, [2]) der Intel-Prozessoren. Diese Befehle hielten Schritt für Schritt mit Pentium MMX, Pentium III und 4 Einzug. Intel unterscheidet MMX (Multimedia Extensions), SSE (Streaming SIMD Extensions), SSE2 und SSE3. Das Prinzip: Ein SIMD-Befehl veranlasst die CPU mehrere Daten gleichen Typs gleichzeitig zu bearbeiten (Abbildung 4). Für normale Applikationen können die Erweiterungen erhebliche Steigerungen der Geschwindigkeit bringen.

Zwar betreffen diese Erweiterungen zu einem sehr großen Teil die Fließkomma-Arithmetik, doch gibt es in beinahe jedem Programm Schleifen, die der Compiler vektorisieren kann. Außerdem kann der Übersetzer durch Analyse des Datenstroms mehrere Speicherzugriffe etwa zu 128 Bit breiten Datentransfers zusammenfassen, die SSE bietet. Berichte von Benutzern und auch die Beobachtungen des Autors belegen, dass dabei der Vektorisierer des ICC dem des GNU-Compilers überlegen ist.

Eigentlich sollten diese Befehlssätze dem Intel-Compiler auch beim Linux-Kernel einen Vorsprung verschaffen können – aber nein. Hintergrund: Die CPU-Registersätze der neuen Instruktionseinheiten decken sich mit den MMX-Fließkommaregistern. Bei jedem Kontextwechsel sichert der Linux-Kernel neben den allgemeinen Registern auch brav die Fließkomma-/MMX- und SSE-Register, sodass alle Prozesse in den Genuss der Erweiterungen kommen.

Findet jedoch ein Wechsel vom Benutzer- in den Kernelmodus statt, so sichern die Einsprungspunkte nur die allgemeinen Register. Sobald nun auch der Programmcode im Kern die Prozessorerweiterungen benutzt, ohne die entsprechenden Register zu sichern, überschreibt der Kernelcode ungefragt Applikationswerte. Und das ist leider das Knock-out für den SIMD-Code im Kern.

Die Funktionen »kernel_fpu_begin« und »kernel_fpu_end«, die einen SIMD-Bereich gegen Präemption schützen und den kompletten FPU/SSE-Kontext sichern, helfen hier nicht weiter. Die SIMD-Codegenerierung des Intel-Compilers erzeugt nämlich an beliebigen Stellen im Kern SIMD-Codesequenzen, und zwar ohne diesen Schutz. Aus der Traum.

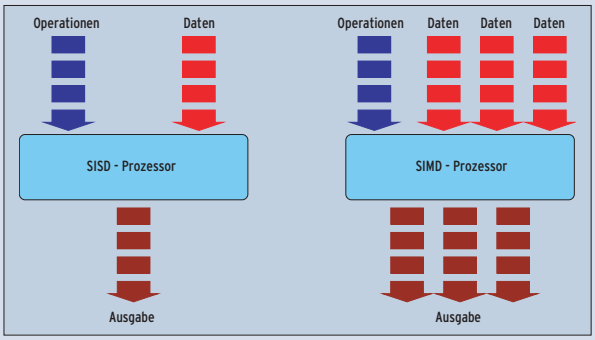


Abbildung 4: Ein SISD-Prozessor (links) arbeitet pro Befehl mit einem einzelnen Datenfeld, während ein SIMD-Prozessor (rechts) in jedem Schritt mehrere Daten gleichzeitig verarbeitet.

es im Binärformat auf eine ähnliche Plattform überträgt, vermag der Linux-Kern nicht vorhandene CPU-Instruktionen durch gleichwertige, kompatible zu ersetzen. Derartige Module erkennt man an den ELF-Sektionen ».altinstr_replacement« und ».altinstructions«.

In »include/asm-i386/processor.h« findet sich die Ursache für die Ausnahme: Die Funktion »prefetch« ist zwar als »extern inline« erklärt, jedoch mit Funktionsrumpf spezifiziert. Ein Wechsel auf »static inline«, was auch eher der praktischen Vernunft entspricht, erzeugt die gewollten Relokationen. Dennoch ist der Grund für das Fehlverhalten des Intel-Compilers an dieser Stelle nicht nachvollziehbar.

Misch-Matsch

Bei diesem Stand der Dinge erleichtert durchatmen wäre verfrüht. Denn wer einen so erzeugten Kernel benutzt, stolpert zügig auf das nächste Problem zu: Das Ext-2-Subsystem versagt beim Abkoppeln (Unmount) einer Partition. Der Grund entpuppt sich als das Hauptproblem des Intel-Compilers: Er kämpft bei gemischten C/Assembler-Blöcken oft mit Problemen und erzeugt dann nachweislich falschen Objektcode.

Dass dieses Fehlverhalten nicht für den Linux-Kern singulär ist, erweist sich zum Beispiel bei der Übersetzung der OpenSSL-Bibliothek 0.9.7d. Deren Selbsttests nach dem Kompilieren mit dem ICC melden Fehler. Gründliches Suchen fördert in diesen Fällen einen Codeabschnitt zutage, der den symmetrischen Verschlüsselungsalgorithmus RC5 implementiert. Zum bitweisen Rotieren verwendet er Inline-Assembler – und schon geht's schief.

Es sieht so aus, dass nach Intel-Lesart Assembler in C-Code nichts zu suchen hat und jeder Entwickler gefälligst auf die Intrinsic-Funktionen des Übersetzers zurückgreifen soll, die Mischcodes vermeiden. Tatsächlich hilft dem RC5-Algorithmus von OpenSSL ein Wechsel auf die Intrinsic-Funktionen »_lrotl« und »_lrotr« auf die Füße.

Tricks'n Traps

Was bei der OpenSSL-Bibliothek noch funktioniert, bringt den Linux-Kernel kaum weiter. Dessen Nähe zur Hardware macht vielerorts C/Assembler-Mischcode unverzichtbar. Und der Intel-Compiler offeriert zu deren Substituierung keine Intrinsic-Funktionen. Die langwierige Suche nach dem erwähnten Fehler im Ext-2-Subsystem über mehrere Kernelkomponenten hinweg fand irgendwann in der Funktion »atomic_dec_and_lock()« ihr Ende.

Listing 1 zeigt den betreffenden Auszug aus der C-Datei »arch/i386/lib/dec_and_lock.c«, den per Intel-Compiler fehlerhaft übersetzte Assembler-Code Listing 2. Der Intel-Compiler analysiert die Inline-Assembler-Parameter falsch und führt den Befehl »cmpxchgl« statt auf

dem eigentlichen Zähler auf einer lokalen Kopie von »atomic->counter« aus. Das ist ein im engeren Sinne kritischer Fehler, zumal dadurch der synchronisierte Zugriff auf den Zähler auf SMP-Rechnern nicht mehr gewährleistet ist. Abhilfe: Sobald das Symbol »__INTEL_COMPILER« definiert ist, wird der kritische Bereich der Funktion »atomic_dec_and_lock« durch komplett neuen Assembler-Code ersetzt.

Vorläufig fertig: Auf den drei Testsystemen des Autors verrichten nach diesem Schema ICC-übersetzte Kernel klaglos ihren Dienst:

- Auf einem Pentium-4-Notebook mit 2 GHz und 512 MByte RAM,
- einem Doppel-Pentium-III-Server mit 2 x 500 MHz und 512 MByte RAM und
- einem Pentium-4-Desktop mit 2,26 GHz und 1024 MByte RAM.

Interprozedurales

Der große Performancevorteil gegenüber GCC-übersetzten Kernen ist aber noch nicht spürbar. Das liegt daran, dass die beiden wichtigsten Mechanismen des Intel-Compilers noch nicht zur Geltung kommen: IPO (Interprocedural Optimization) und PGO (Profile Guided Optimization). Im Zuge der interprozeduralen Optimierung wägt der Intel-Compiler beispielsweise ab, welche Funktionen er als Inlines erweitert und ob es sich für den Ausführungsfluss lohnt, Zwischenresultate in CPU-Registern über Aufrufe hinweg zu propagieren. Festgeschriebene Heuristiken auf Basis der selektierten Prozessorarchitektur treffen dabei die jeweilige Entscheidung [1].

Die Schalterkombination aus »-ipo« und »-ipo_obj« aktiviert den IPO-Mechanismus über alle Quellcodes bis zum Bindungszeitpunkt. Dies fördert dann ein weiteres Problem mit C/Assembler-Mischcodes zutage: In der Datei »include/asm-i386/byteorder.h« quittiert der ICC die Übersetzung mit dem internen Übersetzerfehler »0_(4900 + 5)«. Abhilfe bringt, die Inline-Assembler-Instruktionen durch Intrinsic-Funktionen auszutauschen.

IPO und ihre Tücken

Aber auch die IPO birgt Risiken. Der GNU-Compiler bietet zu dem Schlüsselwort »inline« das Funktionsattribut »always_inline« an, das auch der Intel-Compiler implementiert. Die Kernelsourcen verwenden das Attribut bei allen Inline-Funktionen. Jeder Compiler müsste so gekennzeichnete Funktionen zwingend expandieren, auch wenn das seinem Optimierer gegen den Strich geht. Offenbar sieht der Intel-Compiler das Attribut aber nur als Hinweis an und verlässt sich stattdessen auf seine IPO-Entscheidungen. Dass dieses Vorgehen keine gute Design-Entscheidung der Intel-Entwickler war, wird an den vielen Bitmanipulations-Operationen des Kerns beispielhaft deutlich (definiert in »include/asm-i386/bitops.h«).

Ein Blick in »System.map« verrät, dass der Intel-Compiler viele Vorkommen der Funktionen als eigenständige (statische) Funktionen erzeugt, nicht jedoch als Inlines. Es kann kaum im Sinne der Intel-Entwickler gewesen sein, dass der Übersetzer eine Inline-Funktion, die aus einer Instruktion besteht, zu einer eigenständigen Funktion mit Funktionsprolog und -epilog zuzüglich Aufruf und Stapelbereinigung aufbläht.

Die nahe liegende Lösung, den Übersetzer zur Inline-Expandierung durch Umformulierung der Inline-Funktionen in C-Makros zu forcieren, bekämpft der Intel-Compiler mit allen Mitteln. Denn nur einige betroffene Inline-Funktionen lassen sich umwandeln, ohne dass er wieder falschen Objektcode erzeugt. Hier muss Intel fraglos nachbessern.

Wenn man alle Bitmanipulations-Funktionen als Makros erklären wollte, zöge das eine Spur der Code-Zerstörung durch den Kern. Das vorgestellte Werkzeug »ccd« hilft bei der Lokalisierung dieses Fehlverhaltens. Der Schalter »--assem« veranlasst es dazu, die intermediären Assembler-Codes als Dateien mit der Endung ».c.S« zur Inspektion beizubehalten.

Dreamteam: IPO und PGO

Wenn man nun IPO mit der profilgeleiteten ICC-Optimierung kombiniert, entsteht der gesuchte Geschwindigkeitsvorteil gegenüber dem GNU-Compiler, selbst in der aktuellen Version 3.3.3. In einem dreistufigen Übersetzungsschema greift der ICC nämlich auf Informationen über den tatsächlichen Ausführungsfluss zurück, um Befehle umzuordnen und anderweitig zu optimieren [1].

Dazu übersetzt man das zu optimierende Programm mit aktivierter PGO-Instrumentalisierung und lässt es unter Realbedingungen eine Zeit lang laufen. Der Intel-Compiler erzeugt dabei so genannte PGO Segment Packets und zusätzlichen PGO-Code, der ein Profil des laufenden Programms enthält und die PGO-Segmente zeitgesteuert in Dateien ablegt. In der dritten Phase, der Feedback Compilation, versorgt man den Compiler mit diesen Daten, der somit zielgerichtet übersetzen kann.

Das im vorgestellten Patch enthaltene Kernelmodul »intlpgo« macht den Linux-Kernel PGO-fähig. Es legt dank Interaktion mit dem ebenfalls neuen Daemon »pgod« die Profildaten in Dateien ab. Weiter gehende Informationen zur PGO-Instrumentalisierung des Kerns und zur Erstellung der beiden Programme liefern

Listing 1: Auszug aus »dec_and_lock.c«

```
01 int atomic_dec_and_lock(atomic_t *atomic,
   spinlock_t *lock)
02 {
03         int counter;
04         int newcount;
05
06 repeat:
07         counter = atomic_read(atomic);
08         newcount = counter-1;
09
10         if (!newcount)
11                 goto slow_path;
12
13         asm volatile("lock; cmpxchgl %1,%2"
14                 :"=a" (newcount)
15                 :"r" (newcount), "m" (atomic-
   >counter), "0" (counter));
16
17         /* If the above failed, "eax" will have
   changed */
18         if (newcount != counter)
19                 goto repeat;
20         return 0;
21
22 slow_path:
23         spin_lock(lock);
24         if (atomic_dec_and_test(atomic))
25                 return 1;
26         spin_unlock(lock);
27         return 0;
28 }
```

die Manpages und der Quelltext der beiden Werkzeuge. [5] Die profilgeleitete Optimierung ermöglicht:

- Das Ausführungsverhalten von Kerneln zu analysieren,
- zielgerichtet zu optimieren und
- Spezialkernel anzufertigen.

Abbildung 5 liefert ein Beispiel für den letzten Punkt. Das Intel-Werkzeug »codecov« erzeugt aus den gewonnenen Profildaten für alle Quelltexte HTML-Seiten, die Auskunft über ausgeführte Codeblöcke und Ausführungshäufigkeiten geben. Sofern man den Anwendungsbereich eines vorliegenden Kerns auf bestimmte Szenarien eingrenzen kann, zum Beispiel Desktoprechner, Web- oder Datenbankserver, ermöglicht es der PGO-Mechanismus, darauf spezialisierte Kerne optimiert zu übersetzen. Dies bezieht auch die Ausführungshäufigkeiten der Gerätetreiber real gegebener Hardware mit ein.

Wie viel bringt's?

Reproduzierbare Laufzeitmessungen an übersetzten Kernels gestalten sich relativ schwierig. Linus Torvalds selbst bedient sich zum Optimieren des synthetischen Benchmarks LMBench [6]. Dieser liefert jedoch zu grobkörnige Resultate, um die Kerne seriös zu vergleichen. Das ändert sich, wenn man das Kernelmodul OProfile [7] mit ins Spiel bringt. Es benutzt spezielle Register des Pentium 4, die eine feingranulare Leistungsmessung auf der Ebene der Prozessor-Taktzyklen zulassen.

Um die Wirksamkeit der in diesem Artikel vorgestellten Maßnahmen quantitativ zu belegen, kam folgendes Setup zum Einsatz:

- Pentium-4-Rechner mit 2,26 GHz und 1024 MByte RAM
- LMBench und OProfile zur Messung
- Mit GCC 3.3.3 übersetzter Standardkernel 2.6.5
- Mit ICC 8.0.055 unter Zuhilfenahme der Patches und Tools aus [3] übersetzter Standardkernel 2.6.5 ▶

Listing 2: Assembler »atomic_dec_and_lock«

```
01         .globl atomic_dec_and_lock
02 atomic_dec_and_lock:
03 # parameter 1: 28 + %esp (atomic)
04 # parameter 2: 32 + %esp (lock)
05 ..B1.1:
06   pushl     %esi
07   pushl     %ebx
08   subl      $16, %esp
09   movl      28(%esp), %ebx      # %ebx = atomic
10 ..B1.2:
11   movl      (%ebx), %esi        # %esi = atomic->counter
12   movl      %esi, (%esp)        # (%esp) = counter
13   addl      $-1, %esi           # %esi = newcount
14   je        ..B1.5
15 ..B1.3:
16   movl      (%ebx), %ecx        # %ecx = atomic->counter
17   movl      (%esp), %edx        # %edx = counter
18   movl      %ecx, 12(%esp)      # 12(%esp) = DUPLIKAT VON
19                                  # atomic->counter
20   movl      %edx, %eax          # %eax = counter
21                                  # inline param: "0" (counter)
22 # Begin ASM
23   lock; cmpxchgl %esi,12(%esp)  # %1 = %esi = newcount
24                                  # %2 = 12(%esp) = DUPLIKAT VON
25                                  # atomic->counter
26 # End ASM
27   cmpl      %edx, %eax
28   jne       ..B1.2
29 ..B1.4:
30   xorl      %eax, %eax
31   addl      $16, %esp
32   popl      %ebx
33   popl      %esi
34   ret
35 ..B1.5:
```

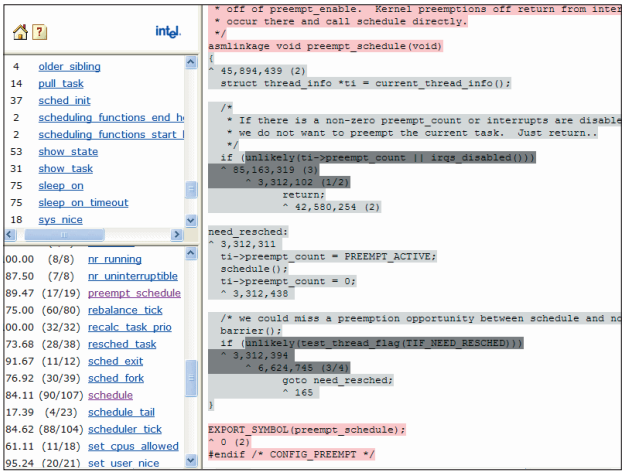


Abbildung 5: Die »preempt_schedule« aus »sched.c« gehört zu den stärker frequentierten Funktionen des Linux-Kerns. Die Häufigkeiten sind an den farbigen Blöcken notiert, die auf ausgeführte, nur partiell ausgeführte und nicht erreichte Blöcke hinweisen.

Tabelle 1: Benchmark-Ergebnisse

LMBench-Test	GCC	ICC	Leistungszuwachs
Datei lesen	667.353	641.146	+3,93 %
Datei lesen (Memmap)	355.139	332.860	+6,27 %
Datentransfer (Pipe)	51.251	46.309	+9,64 %
Datentransfer (TCP/IP-Socket)	404.764	404.371	+0,10 %
Datentransfer (Unix-Socket)	61.422	57.660	+6,12 %
IP-Verbindungslatenz (TCP/IP)	8.727	8.719	+0,09 %
Kontextwechsel	166.269	168.229	-1,18 %
Dateisystem (Erstellen/Löschen)	618.093	611.237	+1,11 %
Datei (Memmap/-unmap)	77.508	70.242	+9,37 %
Seitenfehler (Datei)	1.409	1.351	+4,12 %
IPC-Latenz (Pipe)	31.224	18.243	+41,57 %
Prozesserzeugung	12.832	12.325	+3,95 %
IPC-Latenz (Sun-RPC)	169.344	113.342	+33,07 %
Select (Datei/TCP-Verbindung)	62.676	60.264	+3,85 %
Signale (installieren/auslösen)	53.297	54.053	-1,42 %
Einfacher Systemcall	30.970	30.466	+1,63 %
IPC-Latenz (TCP/IP)	62.500	60.700	+2,88 %
IPC-Latenz (UDP/IP)	58.845	57.982	+1,47 %
IPC-Latenz (Unix-Socket)	54.854	47.464	+13,47 %

- Der Benchmark wurde jeweils neunmal wiederholt, die Ergebnisse arithmetisch gemittelt.

Der profilgeleitet optimierte Kern wurde im Zuge der PGO-Instrumentalisierung verschiedenartigen, aber unspezifischen Belastungen (Dateisystem, Netzwerkverkehr, Hintergrund- und Vordergrundaktivitäten) ausgesetzt. Tabelle 1 zeigt die gemittelten Messungen des Moduls OProfile als Anzahl der Samples, die der NMI (Non Maskable Interrupt) während des Benchmarklaufs im Kernelbereich sammelte. Das ist die Anzahl der Taktzyklen im ungestoppten Zustand bei einem Zählerüberlauf von 22 600 auf dem Testsystem, was bei 2,26 GHz Taktfrequenz einer Zählerauflösung von zirka 10 Mikrosekunden entspricht.

Im Mittel ergibt sich ein Geschwindigkeitsschub von 8,7 Prozent für den Intel-Kern. Das mag im ersten Moment nach wenig klingen, bedeutet rechnerisch jedoch, dass allein durch Neukompilieren stark ausgelastete Maschinen 8,7 Prozent länger arbeiten können, bis sie durch leistungsstärkere ausgetauscht werden müssen. So gesehen besitzt Performancesteigerung auch eine wirtschaftliche Dimension – gerade bei Servern und für Computing-Cluster.

Bemerkenswerte Leistungsspitzen lieferten die Tests »lat_unix« (Interprozess-Kommunikation über Unix-Sockets) mit über 13 Prozent, »lat_rpc« (Kommunikation über Sun-RPC) mit 33 Prozent sowie »lat_pipe« (Kommunikation durch Pipes) mit 41 Prozent. Die beiden einzigen Tests, in denen der Intel-Kern unterlegen war, sind »lat_sig« (Signal-Handler installieren, Signale auslösen) mit minus 1.4 Prozent sowie »lat_ctx« (Kontextwechsel) mit minus 1,2 Prozent.

Ersteres mag daran liegen, dass der Signalcode nicht so häufig durchlaufen wird – eine schlechte Voraussetzung für PGO. Der Wert beim Kontextwechsel kann eine Folge der Basiszeiger sein, die der Kern beim Übersetzen von »sched.c« erfordert. Bei alle Ergebnissen sind Messungenauigkeiten von einem Prozentpunkt oder besser anzunehmen.

Generell vorteilhaft für dedizierte Aufgaben

Die Allgemeingültigkeit des Vergleichs steht durch den PGO-Mechanismus außer Frage, was gleichzeitig die Leistung des Intel-Compilers unterstreicht. Die Ergebnisse erwecken den Eindruck, dass es derzeit offenbar keinen mit dem GCC übersetzten Kernel gibt, der es mit einem nach seinem Einsatzgebiet profilierten, PGO-instrumentalisierten ICC-Kernel aufnehmen könnte.

Zum Erzeugen performanter Linux-Umgebungen für dedizierte Aufgaben, beispielsweise als Datenbankserver, ist anzuraten, alle wichtigen Komponenten (Kern, Server und Bibliotheken) mit dem Intel-Compiler und den Optionen IPO und PGO zu übersetzen und die Maschine eine ausreichend lange Zeit mit realer Last unter der PGO-Instrumentalisierung arbeiten zu lassen. Die Feedback-Übersetzung liefert dann das derzeit schnellst mögliche Gespann aus Kern und Anwendung. *(jk)* ■

Infos

[1] Stephan Siemen, „C- und C++-Programme mit Intels Compiler übersetzen“: Linux Magazin 06/03, S. 58
[2] Stephan Siemen, „Programme für SSE2-fähige Prozessoren optimieren“: Linux Magazin 08/03, S. 101
[3] Intel C-Compiler für Linux: [http://support.intel.com/support/performancetools/c/linux/]
[4] Intel-Dokument über Compiler-Kompatibilität: [http://developer.intel.com/software/products/compilers/techtopics/LinuxCompilersCompatibility.htm]
[5] Kernel-Patch und Werkzeuge: [http://www.pyrillion.org/linuxkernelpatch.html]
[6] LMBench: [http://www.bitmover.com/lmbench]
[7] OProfile: [http://oprofile.sourceforge.net]

Der Autor

Dipl.-Inform. Ingo A. Kubbilun arbeitet als freiberuflicher Autor, Berater und Entwickler [www.kubbilun.de]. Seine freie Software stellt er auf [http://www.pyrillion.org] zur Verfügung.