INTRODUCTION TO OPTIMIZING COMPILATION AND THE INTEL COMPILER

Andrey Bokhanko
Intel

About me

- Andrey Bokhanko
- Manager and technical lead for several Intel Compiler teams
- Started to work on optimizing compilers back in 1999
- Before that, worked in Elbrus, in the compiler team
- PhD thesis on… you guessed it – compilers (register allocation)

Scope of this lecture

- Introduction to *optimizing compilation*
- "Optimizing compilation" – what it is? What is it about?
  - Mostly not on compilers themselves
  - What value Intel Compiler adds?
- Not an introduction to compiler construction – 1 hour is definitely **not** enough for this
- A view from industrial perspective
  - Sorry, I'm not very knowledgeable in state of things in academia

What is a compiler?

- "…is a computer program that transforms source code … into another computer language (object code)" (wikipedia)
- An *essential* tool – you can't write computer programs without it
- An "invisible" tool
- Speed of compilation and stability is essential

What is an optimizing compiler?

- Does everything that a non-optimizing compiler should do
- Also, tries to *optimize* a compiled program, while leaving its semantic intact
  - Not as easy as it might seem
- Frankly, is an *optional*, not essential tool
- Might bring valuable competitive advantage to
  - Software developer
  - OS vendor
  - Hardware vendor

What “optimize” means?

- Make user program faster, smaller, consume less power
  - “Traditional” optimizing compilers concentrate on “faster” part
  - “Smaller” is important only for embedded systems
  - Power-efficiency is extremely important in mobile systems. However, tools are in their infancy
  - Compilation for GP uses distinct approaches; there is a trend for them to become more general
- Preserving semantic of a user program is a must

Who creates optimizing compilers?

- This is a complex and expensive task
- Usually, two types of vendors pay for this work
  - Hardware vendors
    - They want user programs to run fast on their hardware
  - OS vendors
    - They want user programs to run fast on their OSes

Benchmarking

- How to measure if a compiler speed-ups user programs?
    - Easy for a single program, single input, single system
    - Not so easy for multiple programs, compilers, OSes, machines
- We want to simulate and measure what users typically do on a system

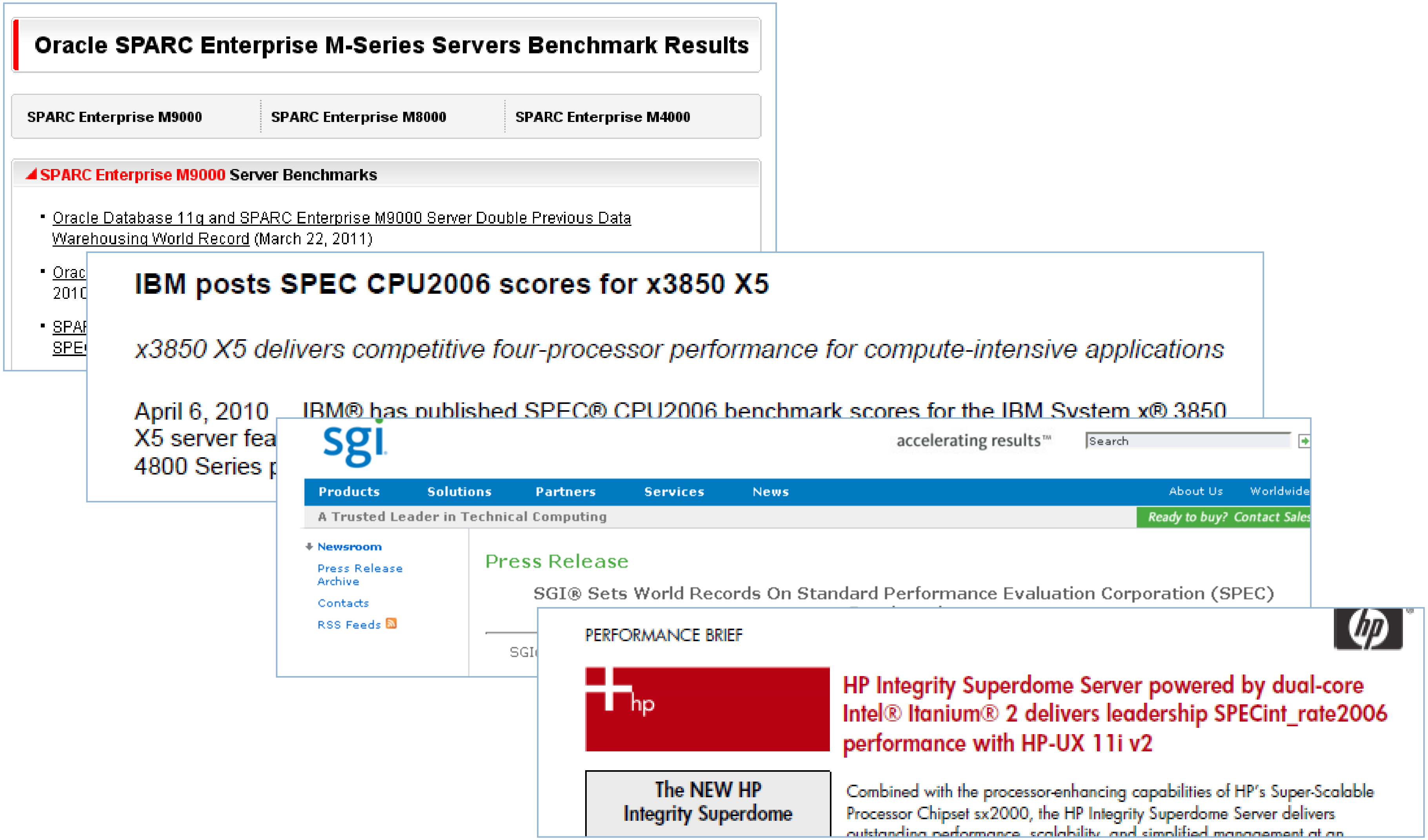
Benchmarking, cont

- Should be done on a set of programs / workloads
  - Representing what customers are most likely going to run on a system
- Should be reproducible
- Overall performance is a combination of
  - Machine
  - OS (including libraries)
  - Compiler

Standard benchmarks

- SPEC: Standard Performance Evaluation Corporation
  - Many benchmarks; most important is SPEC CPU
  - www.spec.org
- TPC: Transaction Processing Performance Council
  - Performance of transaction processing systems (databases)
  - www.tpc.org
- EEMBC: Embedded Microprocessor Benchmark Consortium
  - Benchmarks for embedded/mobile systems
  - Allow some degree of source code modification
  - www.eembc.org
- Kernels, toy programs, synthetic benchmarks (Dhrystone, Whetstone) and MIPS numbers were popular, not anymore

Standard benchmarks, cont



Oracle SPARC Enterprise M-Series Servers Benchmark Results

SPARC Enterprise M9000 Server Benchmarks

- Oracle Database 11g and SPARC Enterprise M9000 Server Double Previous Data Warehousing World Record (March 22, 2011)
- Orac
2010
- SPAR
SPE

IBM posts SPEC CPU2006 scores for x3850 X5

x3850 X5 delivers competitive four-processor performance for compute-intensive applications

April 6, 2010 IBM® has published SPEC® CPU2006 benchmark scores for the IBM System x® 3850
X5 server fea
4800 Series p

Press Release

SGI® Sets World Records On Standard Performance Evaluation Corporation (SPEC)

SGI

PERFORMANCE BRIEF

HP Integrity Superdome Server powered by dual-core Intel® Itanium® 2 delivers leadership SPECint_rate2006 performance with HP-UX 11i v2

Combined with the processor-enhancing capabilities of HP's Super-Scalable Processor Chipset sx2000, the HP Integrity Superdome Server delivers outstanding performance, scalability, and simplified management at an

SPEC CPU

- Probably the most important benchmark for general-purpose computing
  - Though biased towards technical and scientific computing
- First version is SPEC CPU92
- Current version is SPEC CPU2006
- SPECv6 is “almost ready”
- Aims to be vendor- and platform- neutral
  - All major players are members, try to influence SPEC development
- Seriously impacts ASP, especially for server machines
- Makes or breaks careers

What’s inside SPEC CPU2006

- CINT2006
  - 400.perlbench (C, programming language)
  - 401.bzip2 (C, compression)
  - 403.gcc (C, C compiler)
  - 429.mcf (C, combinatorial optimizations)
  - 445.gobmk (C, artificial intelligence: go)
  - 456.hmmer (C, search gene sequence)
  - 458.sjeng (C, artificial intelligence: chess)
  - 462.libquantum (C, physics / quantum computing)
  - 464.h264ref (C, video compression)
  - 471.omnetpp (C++, discret event simulation)
  - 473.astar (C++, path-finding algorithms)
  - 483.xalancbmk (C++, XML processing)

What’s inside SPEC CPU2006, cont

- CFP2006
  - 410.bwaves (Fortran, fluid dynamics)
  - 416.gamess (Fortran, quantum chemistry)
  - 433.milc (C, physics / quantum chromodynamics)
  - 434.zeuscmp (Fortran, physics / CFD)
  - 435.gromacs (C / Fortran, biochemistry)
  - 436.cactusADM (C / Fortran, physics)
  - 437.leslie3d (Fortran, fluid dynamics)
  - 444.namd (C++, biology)
  - 447.dealI (C++, finite element analysis)
  - 450.soplex (C++, linear programming, optimization)
  - 453.povray (C++, image ray-tracing)
  - 454.calculix (C / Fortran, structural mechanics)
  - 459.GemsFDTD (Fortran, computational electronmagnetics)
  - 465.tonto (Fortran, quantum chemistry)
  - 470.lbm (C, fluid dynamics)
  - 481.wrf (C / Fortran, weather)
  - 482.sphinx3 (C, speech recognition)

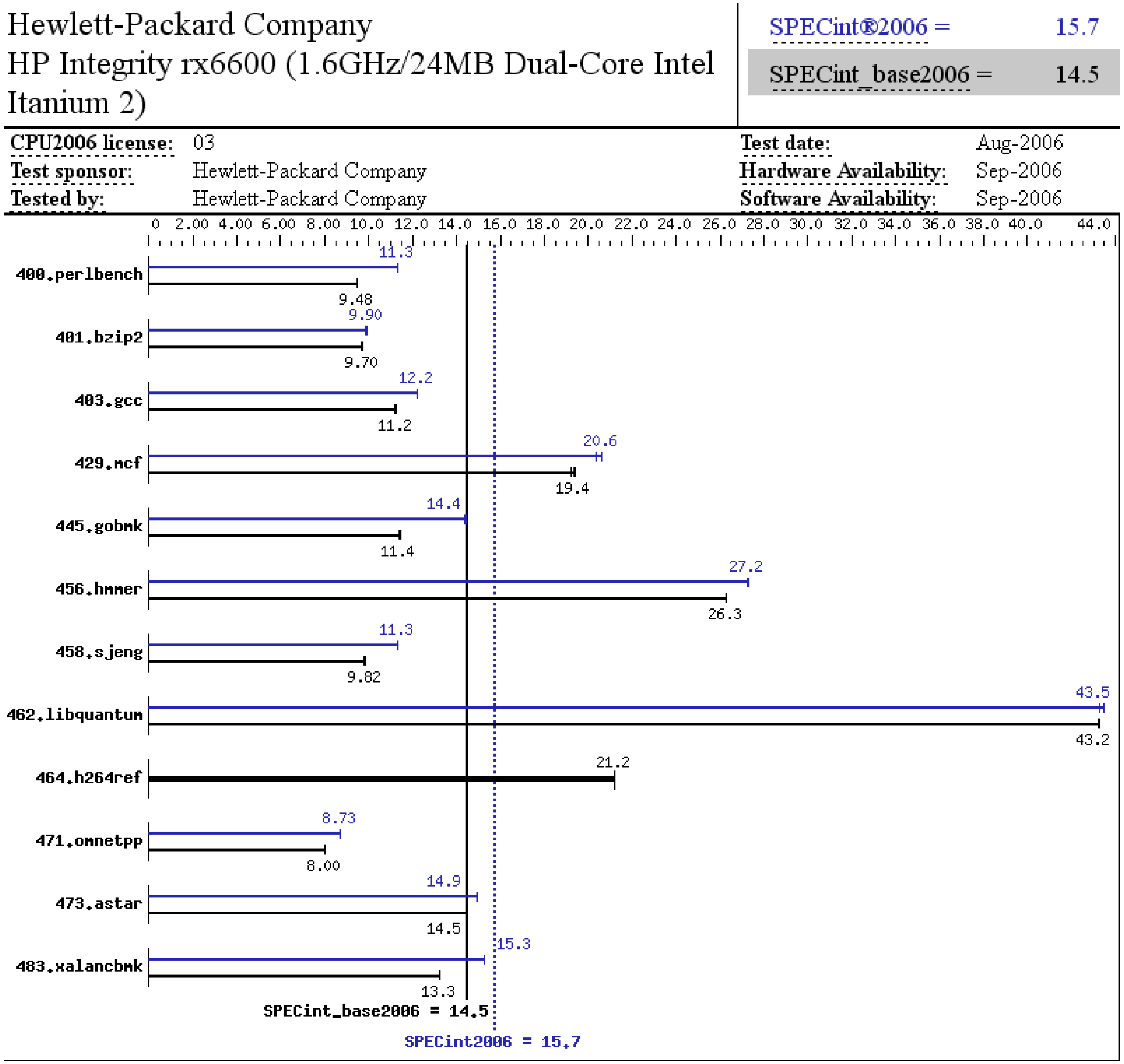
SPEC CPU, cont

- How to measure results?
  - Should we just summarize execution time of all tasks?
- In reality, execution time got normalized to a reference time (obtained on some old machine)

SPEC CPU, cont

- There are two kinds of scores:
  - speed: single copy of each task
  - rate: multiple copies of each task (usually equal to the number of cores)
- Also, two kinds of measurements:
  - base: same options for all tasks
  - peak: different options allowed for different asks
- Total score = geomean of all individual scores
  - Reported separately for CINT and CFP

SPEC CPU, cont

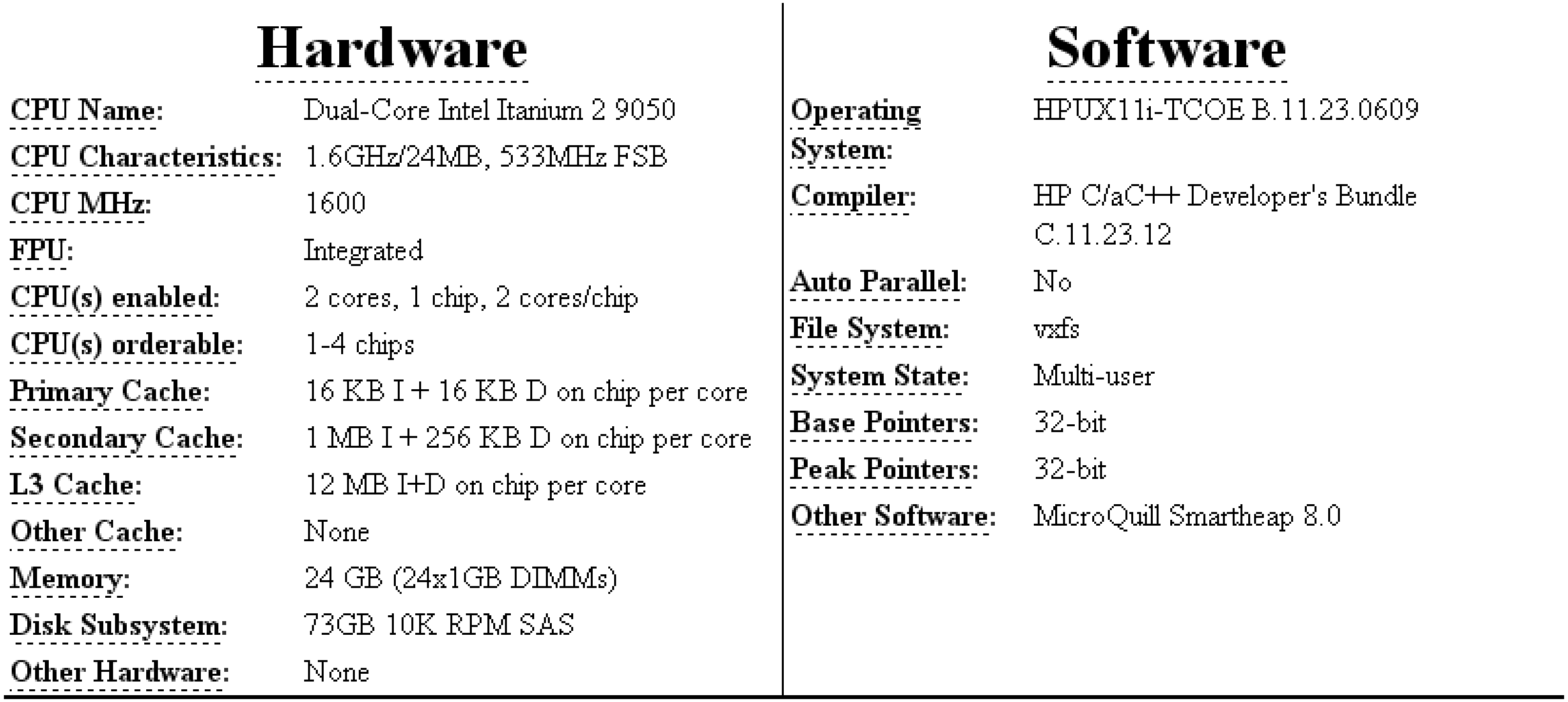


Hewlett-Packard Company
HP Integrity rx6600 (1.6GHz/24MB Dual-Core Intel Itanium 2)

SPECint®2006 = 15.7
SPECint_base2006 = 14.5

CPU2006 license: 03
Test sponsor: Hewlett-Packard Company
Tested by: Hewlett-Packard Company
Test date: Aug-2006
Hardware Availability: Sep-2006
Software Availability: Sep-2006

SPEC CPU, cont



Hardware

CPU Name:	Dual-Core Intel Itanium 2 9050
CPU Characteristics:	1.6GHz/24MB, 533MHz FSB
CPU MHz:	1600
FPU:	Integrated
CPU(s) enabled:	2 cores, 1 chip, 2 cores/chip
CPU(s) orderable:	1-4 chips
Primary Cache:	16 KB I + 16 KB D on chip per core
Secondary Cache:	1 MB I + 256 KB D on chip per core
L3 Cache:	12 MB I+D on chip per core
Other Cache:	None
Memory:	24 GB (24x1GB DIMMs)
Disk Subsystem:	73GB 10K RPM SAS
Other Hardware:	None

Software

Operating System:	HPUX11i-TCOE B.11.23.0609
Compiler:	HP C/aC++ Developer's Bundle C.11.23.12
Auto Parallel:	No
File System:	vxfs
System State:	Multi-user
Base Pointers:	32-bit
Peak Pointers:	32-bit
Other Software:	MicroQuill Smartheap 8.0

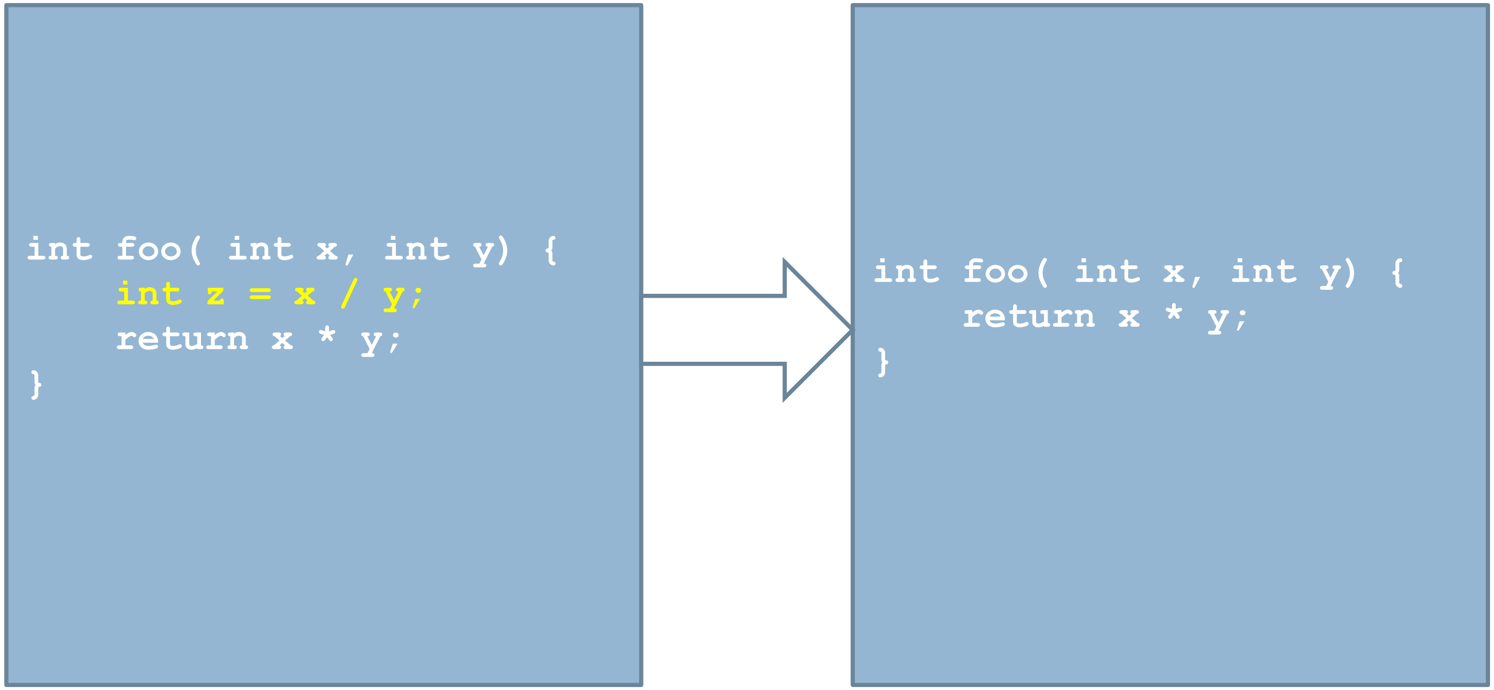
How to optimize?

- Basically, two ways:
- Eliminate redundant / slow computations
  - Classic optimizations
- Keep execution resources busy
  - Especially important for statically-scheduled machines
- Sometimes, these two goals conflict with each other

Elimination of redundant / slow computations

- Most classic optimizations
  - Dead code elimination
  - Common subexpression elimination
  - Constant folding
  - Strength reduction
  - …
- Generally, help everywhere, so implemented everywhere
- Known for very, very long time

Dead code elimination



```
int foo( int x, int y) {
    int z = x / y;
    return x * y;
}
```

```
int foo( int x, int y) {
    return x * y;
}
```

Common subexpression elimination

```
int foo( int x, int y) {
    int z1 = x * y + 1;
    int z2 = x * y + 2;
    return z1 + z2;
}
```

```
int foo( int x, int y) {
    int t = x * y;
    int z1 = t + 1;
    int z2 = t + 2;
    return z1 + z2;
}
```

CSE + constant folding + DCE

```
int foo( int x, int y) {
     int z1 = x * y + 1;
     int z2 = x * y + 2;
     return z1 + z2;
}
```

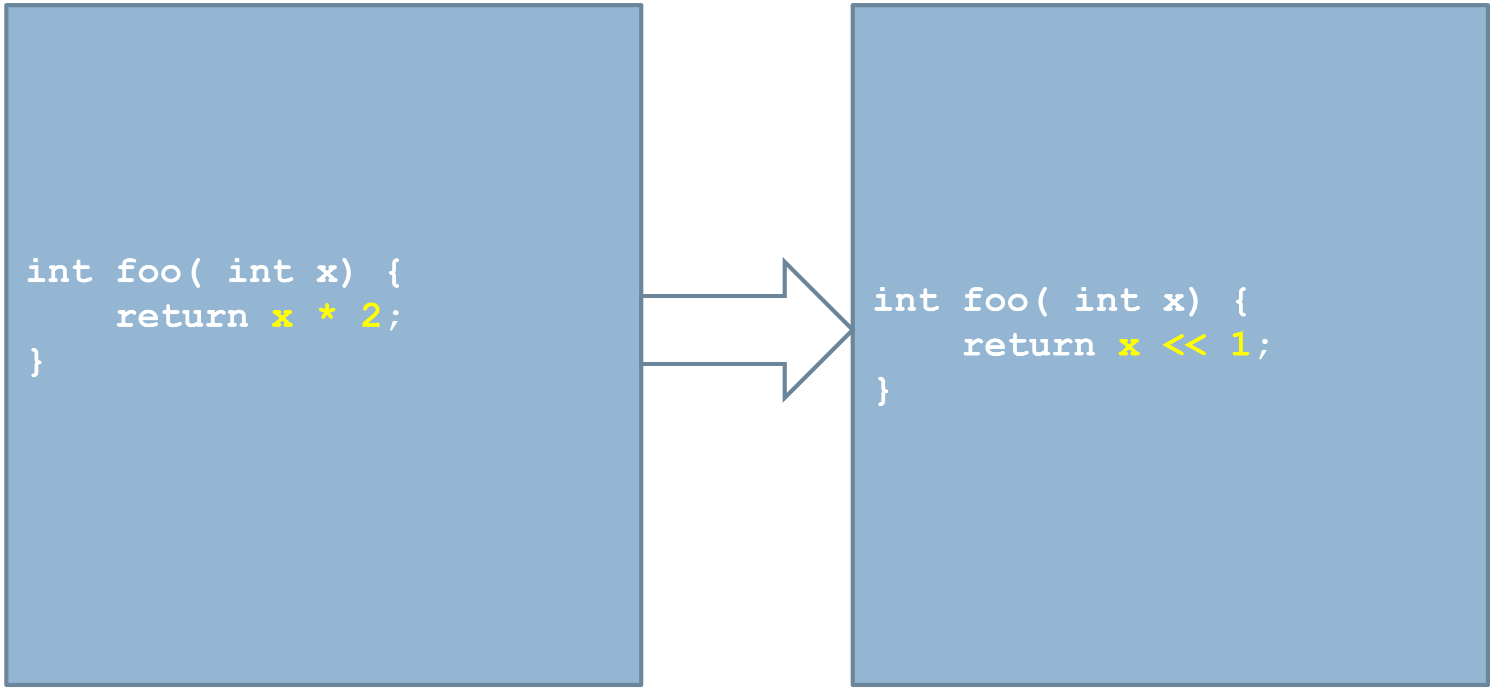
```
int foo( int x, int y) {
     int t = x * y;
     return t + t + 3;
}
```

Strength reduction

```
int foo( char *A) {
  for (int i = 0; i < 100; i++) {
    *((int *)(A + i * 4)) = 0;
  }
}
```

```
int foo( char *A) {
  char *t = A + i;
  for (int i = 0; i < 100; i++) {
    *((int *)t) = 0;
    t += 4;
  }
}
```

Peephole



```
int foo( int x) {
    return x * 2;
}
```

```
int foo( int x) {
    return x << 1;
}
```

Keeping execution resources busy

- When execution resources may lay unused?
  - Parallel machine with only some of available execution resources used
  - Waiting for a dependency
    - Especially memory dependency!
- Advanced, aggressive, speculative scheduling
- Memory optimizations
- Often implemented in hardware, especially in OOO machines

Advanced, aggressive, speculative scheduling

- Scheduling is reordering of instructions in order to keep execution units busy all the time
- Advanced = using advanced techniques, like copying of instructions
- Aggressive = global in scope
- Speculative = executing instructions that might not be executed

Control speculation

```
int foo( int *x) {
    if (x != 0) {
        return *x;
    }

    return 0;
}
```

```
int foo( int *x) {
    int t = *x;
    if (x != 0) {
        return t;
    }

    return 0;
}
```

Data speculation

```
int foo( int *x, int *y) {
    if (x != 0) {
        *y = 0;
        return *x;
    }

    return 0;
}
```

```
int foo( int *x, int *y) {
    int t = *x;
    if (x != 0) {
        *y = 0;
        return t;
    }

    return 0;
}
```

Unrolling

```
int foo( char *A) {
  char *t = A + i;
  for (int i = 0; i < 100; i++) {
    *((int *)t) = 0;
    t += 4;
  }
}
```

```
int foo( char *A) {
  char *t1 = A + i;
  char *t2 = A + i + 4;
  for (int i = 0; i < 50; i++) {
    *((int *)t1) = 0;
    t1 += 8;
    *((int *)t2) = 0;
    t2 += 8;
  }
}
```

Prefetching

```
int foo( int *A) {
  int x = 0;
  for (int i = 0; i < 100; i++) {
    x += A[i];
  }

  return x;
}
```

```
int foo( int *A) {
  int x = 0;
  for (int i = 0; i < 100; i++) {
    x += A[i];
    prefetch A[i + 4];
  }

  return x;
}
```

Profiling

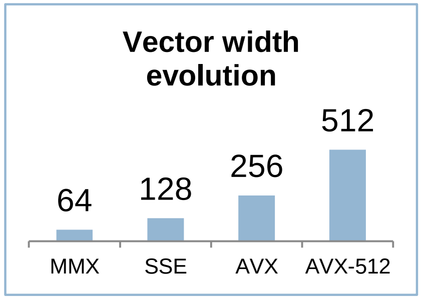
- It is important to know *where* and *how* to optimize
- Instrumentation of user program, then “profile collection” run
- Several deficiencies
  - Transforms compilation into two-step process
  - How to choose input for profile collection run? What if it is not representable?

Inter-procedural (aka link-time) optimizations

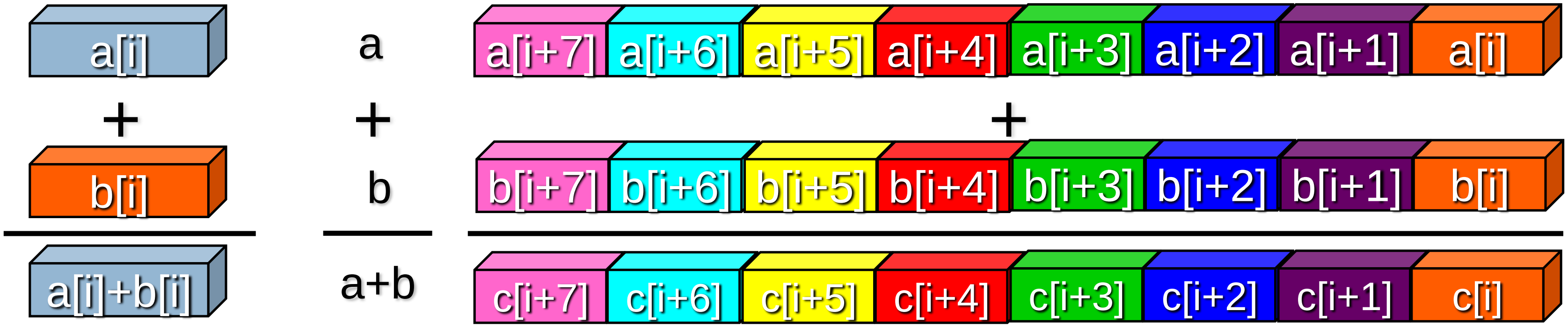
- Optimizations across function (and translation unit) boundaries
  - Inlining
  - Function cloning
  - Interprocedural constant propagation
  - …
- Individual files compiled as usual
- Final linking step does all the optimizations
  - …and usually takes a lot of time!

Vectorization

- Practically all modern processors support SIMD instructions
- **S**ingle **I**nstruction **M**ultiple **D**ata



```
for (i=0;i<=MAX;i++)
    c[i]=a[i]+b[i];
```



SIMD: how to use

Auto

- High and transparent portability
- High and transparent scalability
- No development cost
- Unpredictable performance

```
…
float *a, *b, *c;
…
for(int i…)
  c[i] = a[i] + b[i];
$> icc -fast …
```

Explicit

- Enforces vectorization performance
- High and transparent portability
- High and transparent scalability
- Low development cost
- Predictable performance

```
…
float *a, *b, *c;
…
#pragma omp simd
for(int i…)
  c[i] = a[i] + b[i];
$> icc –fast –openmp …
```

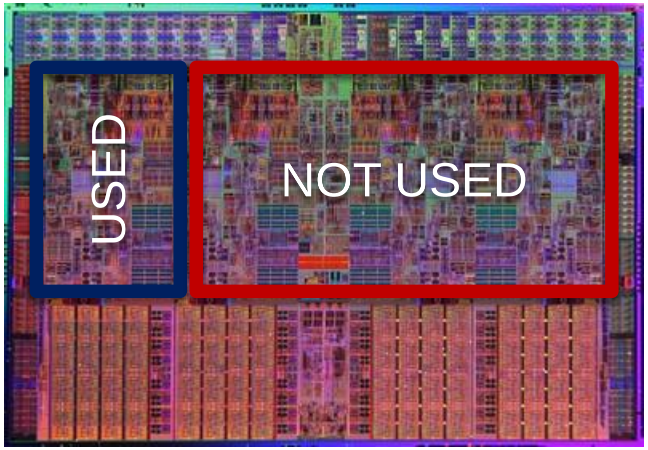
Manual

- Max performance
- Low portability
- High development cost
- Low scalability

```
#include "xmmintrin.h“
…
float *a, *b, *c;
__m128 ma, mc, mc;
…
for(int i…)
{
 ma = _mm_load_ps(a + i*4);
 mb = _mm_load_ps(b + i*4);
 mc = _mm_add_ps(ma, mb);
  _mm_store_ps(c+i * 4, mc);
}
$> icc …
```

Parallelization

- Most modern processors have multiple cores
- If you don’t employ parallelization, all but one core are lost
- Different OS-specific standards: pthreads, Windows threads, Apple blocks
- OpenMP is a platform- and vendor- neutral standard
  - You had a separate lecture on it



What value Intel Compiler adds?

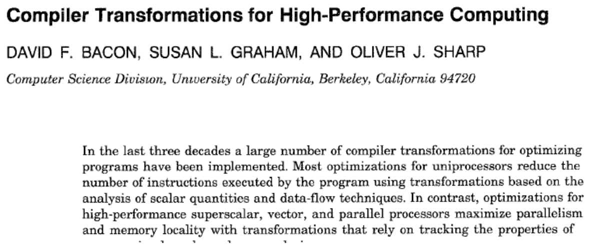
- Focused on delivering maximum performance on IA
    - Powerful loop, profile-based and IPO optimizations
    - Beats other compilers
    - Used by practically everyone to publish SPEC scores on IA
- Intel compiler developers collaborate with Intel HW engineers to implement optimizations
- Supports latest IA instructions
    - Usually much sooner than other compilers

What value Intel Compiler adds?, cont

- Supports latest parallel programming standards
  - OpenMP 4.0
  - CilkPlus
- Broad support for vectorization on IA
  - From manual to automatic, with #pragma omp simd in between
  - Latest SIMD extensions

What to read

- Bacon at al, “Compiler Transformations for high-performance computing”, ACM Computing Surveys, Dec 1994
- Steven Muchnick, “Advanced Compiler Design and Implementation”, Morgan Kaufmann, 1997
- Schouten at al, “Inside the Intel Compiler”, Linux Journal, Feb 2003
- PLDI, CGO conferences

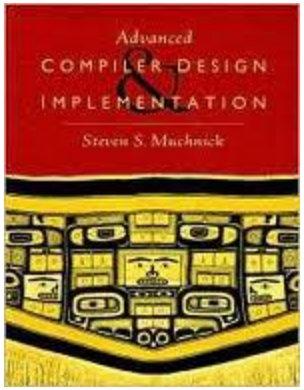


Compiler Transformations for High-Performance Computing

DAVID F. BACON, SUSAN L. GRAHAM, AND OLIVER J. SHARP

Computer Science Division, University of California, Berkeley, California 94720

In the last three decades a large number of compiler transformations for optimizing programs have been implemented. Most optimizations for uniprocessors reduce the number of instructions executed by the program using transformations based on the analysis of scalar quantities and data-flow techniques. In contrast, optimizations for high-performance superscalar, vector, and parallel processors maximize parallelism and memory locality with transformations that rely on tracking the properties of



Thank you for your time!

- Questions?
- andreybokhanko@gmail.com