Konstrukce překladačů

David Bednárek

www.ksi.mff.cuni.cz

Pravidla studia

NSWI109

2/1

Z,Zk

Pravidla studia

- **Cvičení**
  - Každých 14 dní
  - Zápočtové testy
    - Hromadný termín na cvičení koncem semestru
    - Opravné termíny individuálně ve zkušebních termínech
- **Přednáška**
  - Zkouška – ústní s písemnou přípravou

Literatura

Literatura

- A.V. Aho, R. Sethi, J.D. Ullman
  Compiler: Principles, Techniques and Tools (1986, 2007)
- Grune, Bal, Jacobs, Langendoen
  Modern Compiler Design (2000)
  - Přehled včetně front-endů a překladu neprocedurálních jazyků
- Steven S. Muchnick
  Advanced Compiler Design and Implementation (1997)
  - Přehled optimalizací v back-endech
- Randy Allen, Ken Kennedy
  Optimizing Compilers for Modern Architectures (2001)
  - Hardwarově závislé optimalizace
- R. Morgan
  Building an Optimized Compiler (1998)
- Srikant, Shankar (eds.)
  The Compiler Design Handbook (2003)
  – Optimizations and Machine Code Generation
  - Sbírka 22 článků
- J. R. Levine
  Linkers and Loaders (1999)

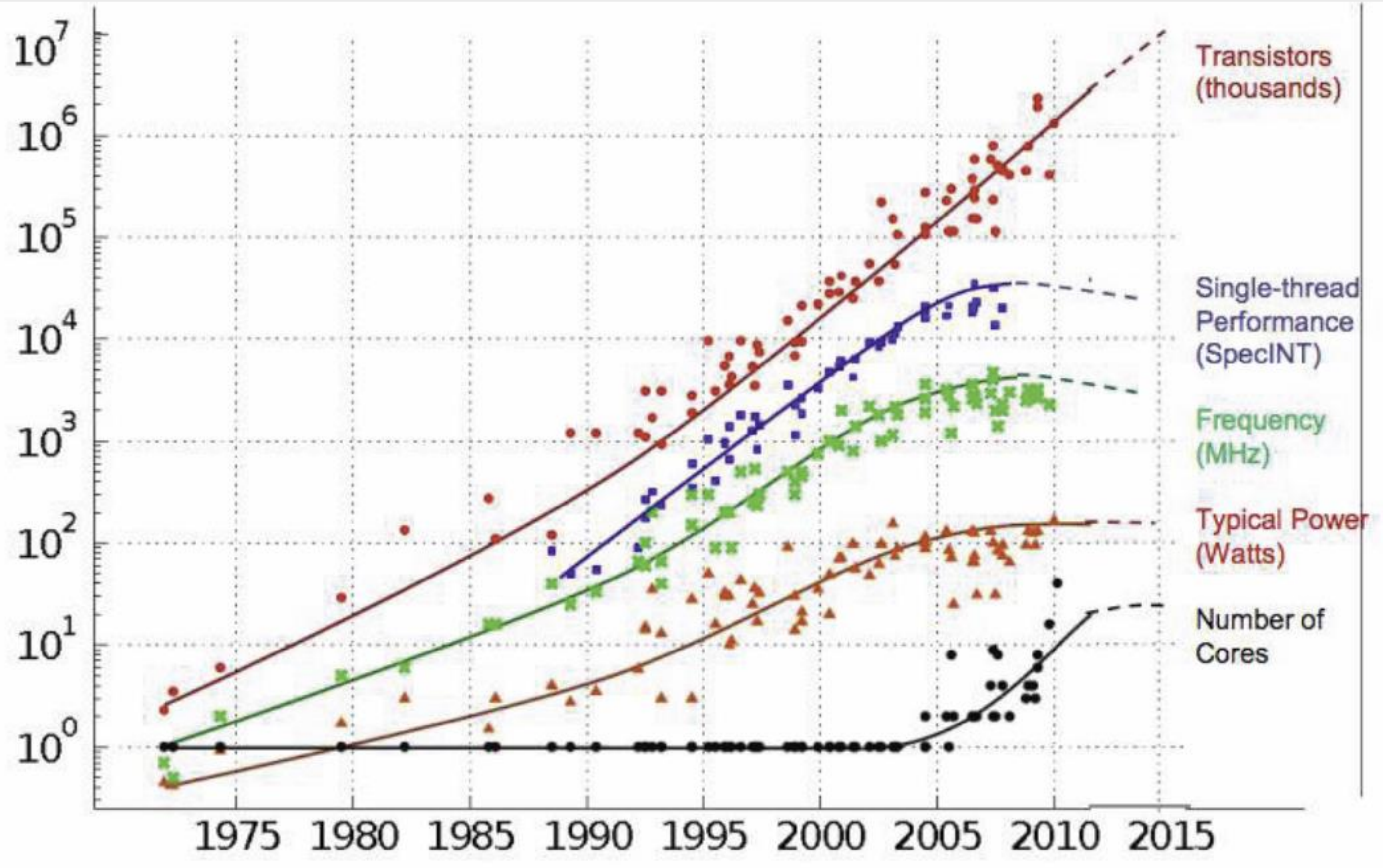
Historie překladačů

Vývoj hardware a programovacích jazyků

- **1957: FORTRAN (IBM)**
  - První překladače
- **1960: IBM 360 - general-purpose registers**
  - Alokace registrů
- **1970: Cray - Pipeline**
  - Scheduling
- **1993: PowerPC (IBM) - Out-of-order execution**
  - Scheduling v HW
- **2008: Intel Atom, ARMv7, GPGPU - In-order execution**
  - Scheduling překladačem opět důležitý

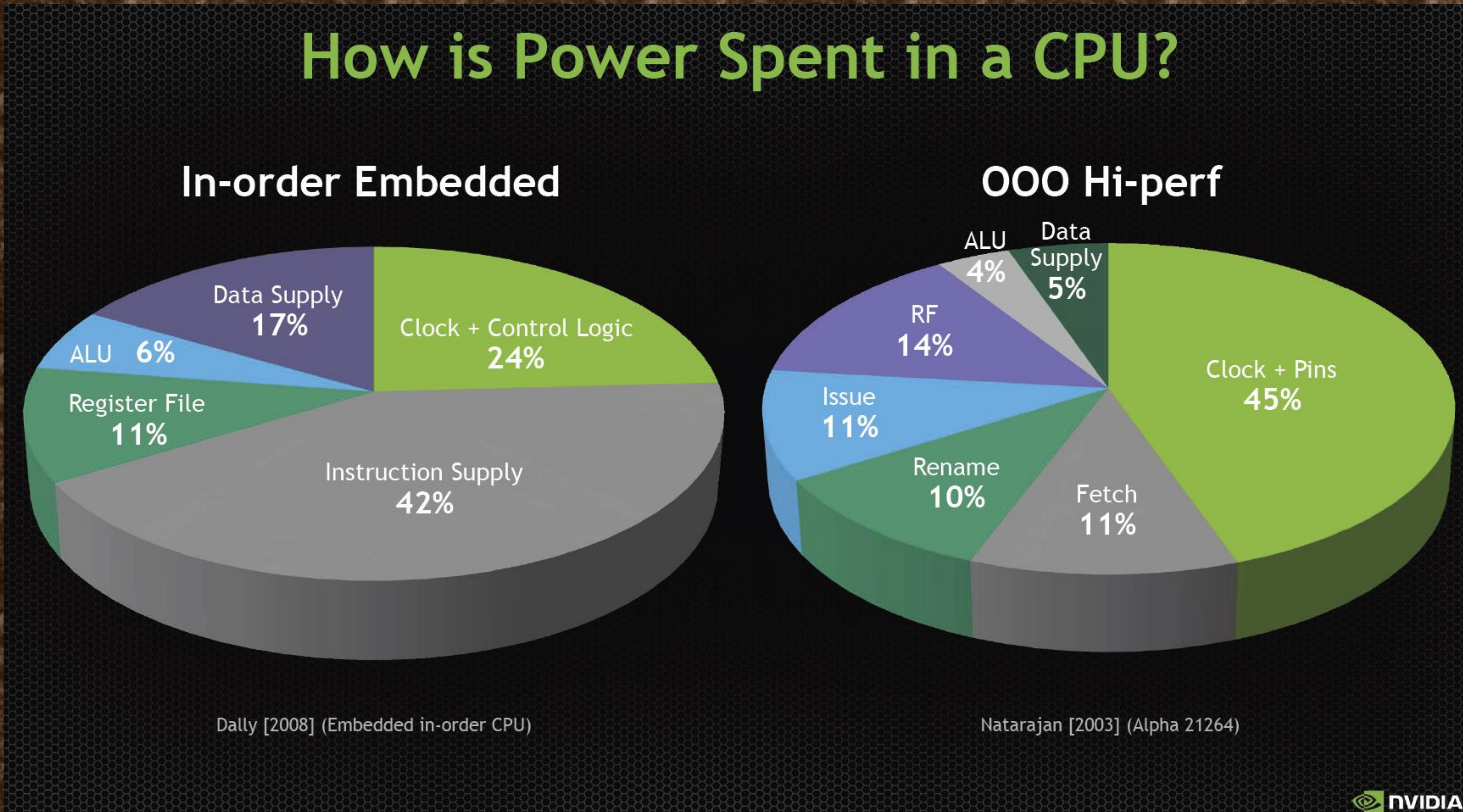
Source: James Laurus @ HiPEAC 2015

Dennard Scaling is Dead

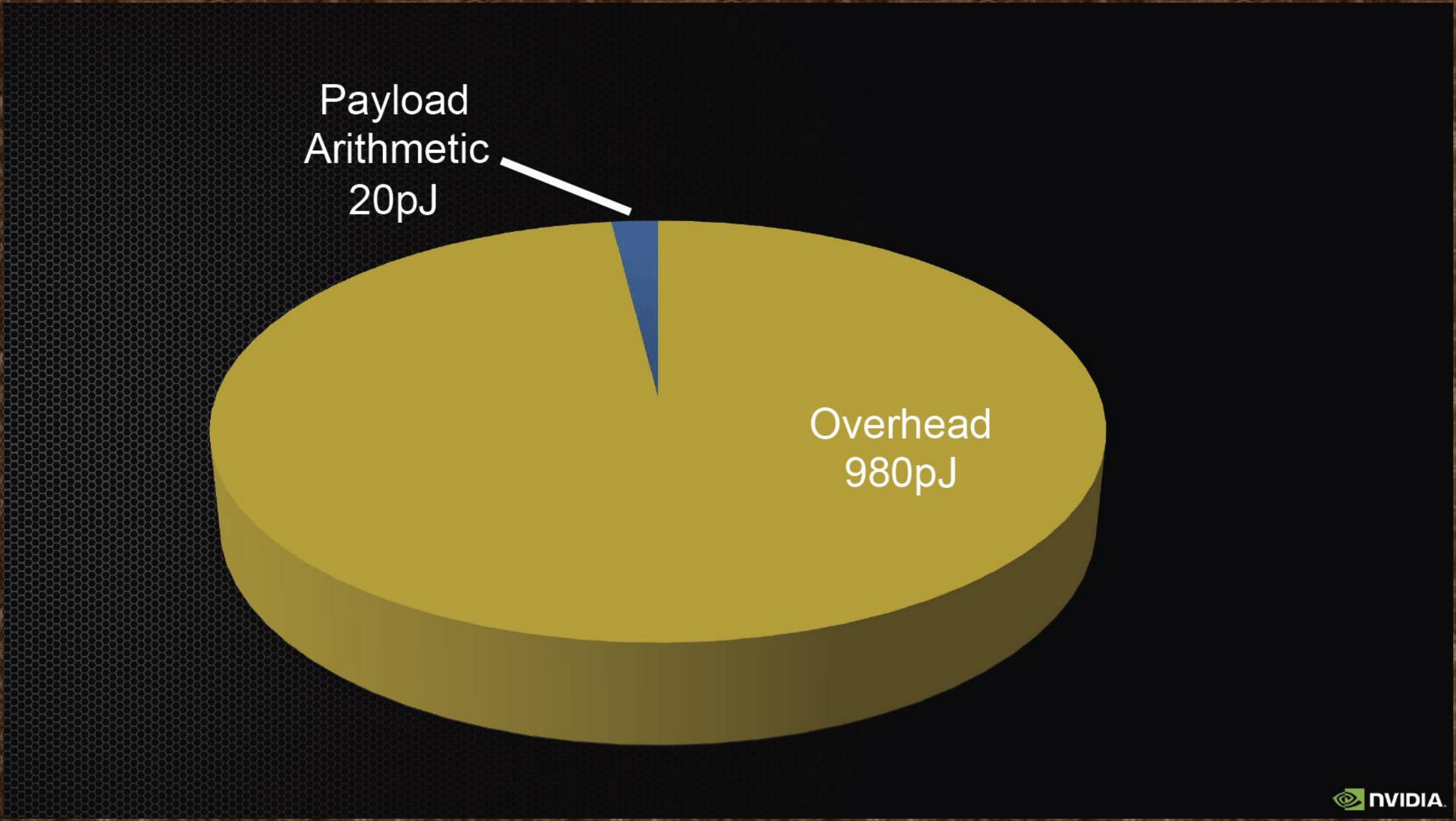


Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

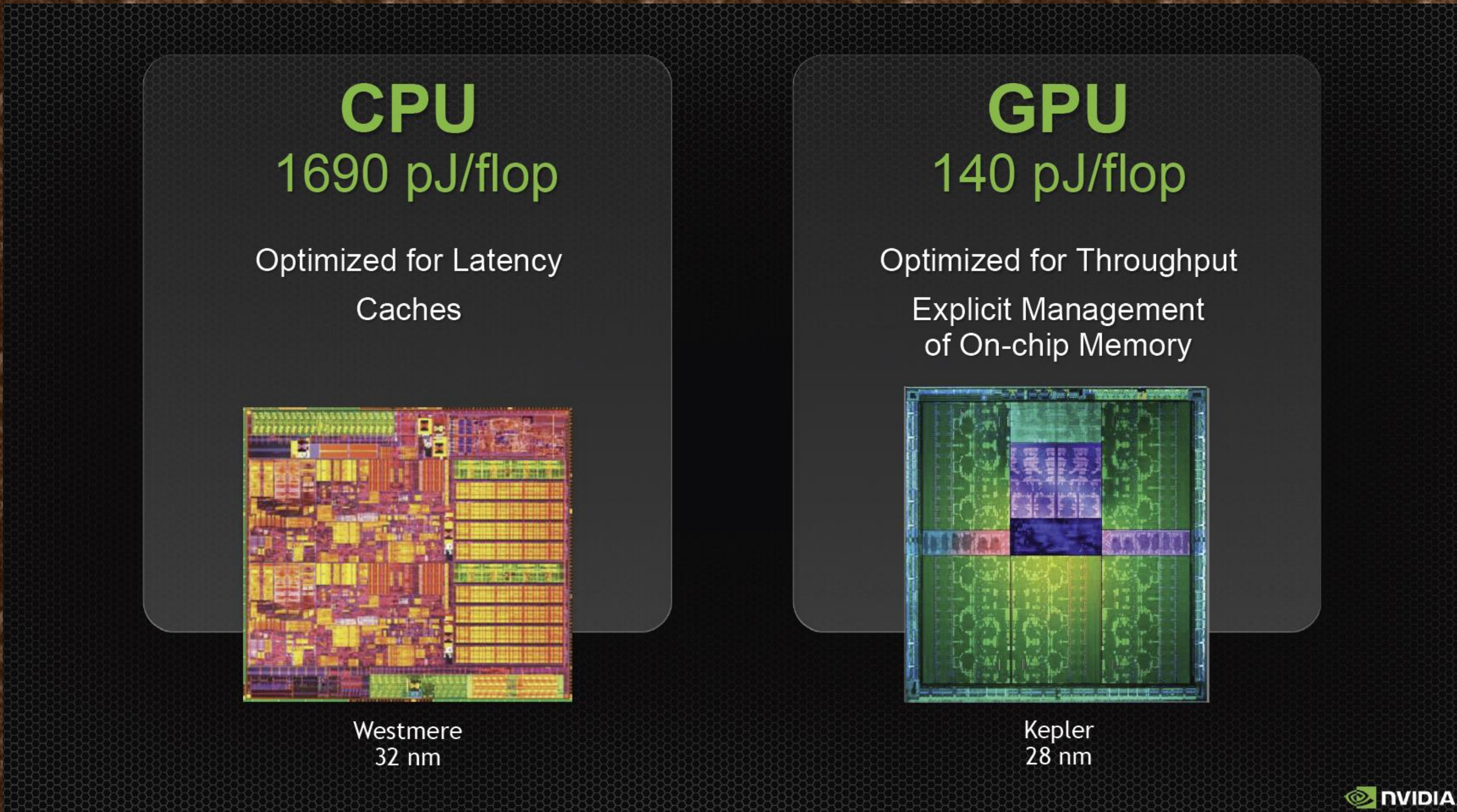
Source: William J. Dally @ HiPEAC 2015



Source: William J. Dally @ HiPEAC 2015



Source: William J. Dally @ HiPEAC 2015



Compilation Scenarios

Compilation Scenarios

- **AOT: Ahead-Of-Time Compilation**
  - FORTRAN, C, C++, many historic languages
  - Enough time to compile
  - Not enough information on the runtime environment
    - Blended code often required
- **JIT: Just-In-Time Compilation**
  - Java, C#, Java(ECMA)Script, (PHP2021)
    - Enforced and limited by the dynamic features of the languages
  - Compile only the most frequently used code
    - Little chance for inter-procedural optimization
  - Precise information on
    - Target architecture
    - Overall workload
    - Compiled program behavior

Compilation Scenarios

- **LTO: Link-Time Optimization**
  - Linking in Intermediate Representation (not target code)
    - Save time when compiling duplicated code
      - C++ inline functions and templates
  - Whole-program analysis and optimization
    - Inter-procedural optimization across modules

- **PDO: Profile-Driven Optimization**
  - Compile the program with instrumentation
    - Additional code producing statistics (profile) at runtime
  - Run the program under “typical” load
    - Slightly slowed by the additional instrumentation code
  - Compile again using the profile
    - Detect frequently used code and control-flow paths
    - Estimate typical array and loop sizes

Notable Compilers

Notable Open-Source Compilers

- **GCC (GNU Compiler Collection) - 1987**
  - Developed alongside Linux
    - Linux kernel interface is de-facto standardized as a part of glibc
    - Use outside of Linux possible but difficult
  - C/C++, FORTRAN, Go, (Objective-C), (Java), ...
  - Support for almost all target platforms in existence
- **Clang/LLVM**
  - LLVM started at University of Illinois – 2000
    - Originally scientific testbed for creating compilers
      - GCC internals were considered obsolete and impenetrable
    - Now includes code generators and optimizers
      - AOT + Experimental JIT support
      - x86, x86-64, ARM, (Nvidia PTX), ...
  - Clang started by Apple – 2005
      - GCC team unwilling to improve Objective-C support
      - Apple hired the LLVM team to create C/Objective-C/C++ front-end

Example

Clang/LLVM

Clang/LLVM 6.0.0 x86-64, -O3 -mno-sse


```
char chksum(char* p, int i)
{
    char s = 0;
    while (i > 0)
    {
        s ^= *p++;
        --i;
    }
    return s;
}
```

```
_Z6chksumPci:
    test esi, esi
    jle  .LBB0_1
    add  esi, 1
    xor  eax, eax
.LBB0_3:
    xor  al, byte ptr [rdi]
    add  rdi, 1
    add  esi, -1
    cmp  esi, 1
    jg   .LBB0_3
    ret
.LBB0_1:
    xor  eax, eax
    ret
```

Clang/LLVM 6.0.0 x86-64, -O3 -mavx2


```
_Z6chksumPci:
    test    esi, esi
    jle     .LBB0_1
    mov     eax, esi
    not     eax
    cmp     eax, -3
    mov     edx, -2
    cmovg   edx, eax
    lea     eax, [rdx + rsi]
    add     eax, 1
    add     rax, 1
    cmp     rax, 128
    jae     .LBB0_4
    xor     eax, eax
    mov     rcx, rdi
    jmp     .LBB0_7
.LBB0_1:
    xor     eax, eax
    ret
.LBB0_4:
    add     edx, esi
    add     edx, 2
    and     edx, 127
    sub     rax, rdx
    sub     esi, eax
    lea     rcx, [rdi + rax]
    add     rdi, 96
    vpxor   xmm0, xmm0, xmm0
    vpxor   xmm1, xmm1, xmm1
    vpxor   xmm2, xmm2, xmm2
    vpxor   xmm3, xmm3, xmm3
```

```
.LBB0_5:
    vpxor   ymm0, ymm0, ymmword ptr [rdi - 96]
    vpxor   ymm1, ymm1, ymmword ptr [rdi - 64]
    vpxor   ymm2, ymm2, ymmword ptr [rdi - 32]
    vpxor   ymm3, ymm3, ymmword ptr [rdi]
    sub     rdi, -128
    add     rax, -128
    jne     .LBB0_5
    vpxor   ymm0, ymm1, ymm0
    vpxor   ymm0, ymm2, ymm0
    vpxor   ymm0, ymm3, ymm0
    vextracti128        xmm1, ymm0, 1
    vpxor   ymm0, ymm0, ymm1
    vpshufd xmm1, xmm0, 78
    vpxor   ymm0, ymm0, ymm1
    vpshufd xmm1, xmm0, 229
    vpxor   ymm0, ymm0, ymm1
    vpsrld  xmm1, xmm0, 16
    vpxor   ymm0, ymm0, ymm1
    vpsrlw  xmm1, xmm0, 8
    vpxor   ymm0, ymm0, ymm1
    vpextrb eax, xmm0, 0
    test    edx, edx
    je      .LBB0_9
.LBB0_7: add esi, 1
.LBB0_8: xor al, byte ptr [rcx]
    add     rcx, 1
    add     esi, -1
    cmp     esi, 1
    jg      .LBB0_8
.LBB0_9:
    vzeroupper
    ret
```

Clang/LLVM 6.0.0 x86-64, -O3 -mno-sse

```c
char chksum(char* p, int i)
{
    char s = 0;
    while (i > 0)
    {
        s ^= *p++;
        --i;
    }
    return s;
}
```

```llvm
define signext i8 @_Z6chksumPci(
    i8* nocapture readonly, i32)
    local_unnamed_addr #0 {
  %3 = icmp sgt i32 %1, 0
  br i1 %3, label %4, label %14
; <label>:4: ; preds = %2
  br label %5
; <label>:5: ; preds = %4, %5
  %6 = phi i8 [ %11, %5 ], [ 0, %4 ]
  %7 = phi i32 [ %12, %5 ], [ %1, %4 ]
  %8 = phi i8* [ %9, %5 ], [ %0, %4 ]
  %9 = getelementptr inbounds i8, i8*
    %8, i64 1
  %10 = load i8, i8* %8, align 1, !tbaa
    !2
  %11 = xor i8 %10, %6
  %12 = add nsw i32 %7, -1
  %13 = icmp sgt i32 %7, 1
  br i1 %13, label %5, label %14
; <label>:14: ; preds = %5, %2
  %15 = phi i8 [ 0, %2 ], [ %11, %5 ]
  ret i8 %15
}
```