



## 4.2.5 BD and CG work

Presented by  
Michael C. Berg



# Overview

- Opportunity analysis goals
  - Discover code patterns
  - Inefficiencies in generated code
  - Categorize missed opportunities
  - Describe prefetching for new model
  - Examine AVX code quality

# Overview

- Opportunity analysis
  - Discovered many performance areas which were applied in 4.2.5 or are planned
    - Scale-Index-Base addressing needed as default addressing mode
    - Vectorization opportunities identified
    - More unrolling could be applied
    - CG based multi-version loops
    - Multi-version loops for alignment
    - Multi-version loops for Interior pointers
    - Outer loop unrolling cases
    - Register pressure based algorithms in CG
    - Partial Vectorization problems
  - New Prefetching model for BD
  - AVX/FMA4 modeling
    - When to use destructive destination and when not
    - When to load-execute and when to not

# Overview

- Implementations
  - BD machine model/info and expansion
  - Multi-version for alignment in memcpy lib routine
  - Interior pointer translation
  - Best fit loop unrolling(now the default)
  - Register pressure utilities, algorithms and heuristics
  - Loop remainder subsuming to inner loops
  - EBO opportunities for Inc/dec
  - Min/Max for vectorization
  - Load execution for Vector operations

# Implementations

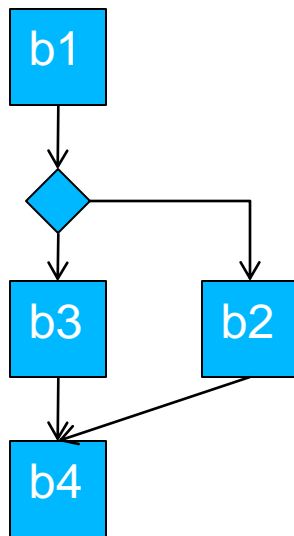
- BD machine model/info and expansion
  - Cache/Prefetch modeling
    - Software prefetch off by default on BD
  - Machine Description
    - Properties, Scheduling info, New instruction operand layout, Assembly formatting
  - CG Dependence map updates for new instructions
  - Auto translation from legacy SSE to AVX in cg expansion
    - Saved dramatically on top code mapping in expansion
    - When ops are created, we simply create the right kind for AVX
  - FMA4 support
    - CG expansion patterns
    - Load execution
    - All forms supported(add, sub, addsub, subadd, net, etc)
    - FMA disassociation

For info about the new architecture see:

[http://support.amd.com/us/Processor\\_TechDocs/47414.pdf](http://support.amd.com/us/Processor_TechDocs/47414.pdf)

# Implementations

- Multi-version for alignment in memcpy lib routine
  - Add flow to pose the default and optimized case
  - Test to Align to granular 128bit reads
  - Utilize aligned loop when test succeeds
  - Utilize standard loop when test fails



Multi-version Loop in CG:  
The original loop prolog is b1,  
the original loop is b2 and the  
optimized loop is b3. The join and  
any fixup is the epilog block, b4.

# Implementations

- Interior pointer translation(with Multi-Version loops)
  - Utilizes discovery of similar address expressions on a collection of variables
  - Answers the question of can these expressions share index resources in SIB form
  - Reduces the number of unique address components needed to provide array accesses

```
DO I = I1, I2
DO J = J1, J2
    DUDZ(I) = DZI * R12I * (U(I, J, K) - U(I, J+1, K) +
    >      8.0D0 * (U(I, J+2, K) - U(I, J+1, K)))
    DVDZ(I) = DZI * R12I * (V(I, J, K) - V(I, J+1, K) +
    >      8.0D0 * (V(I, J+2, K) - V(I, J+1, K)))
    DWDZ(I) = DZI * R12I * (W(I, J, K) - W(I, J+1, K) +
    >      8.0D0 * (W(I, J+2, K) - W(I, J+1, K)))
    END DO
END DO
```

Where arrays U, V and W all share tests to see if the distance between J and J+1, and J+1 and J+2 in the above accesses are the same, hence enabling sharing index address calculation resources for U, V and W. With Multi-Version loops, we also emit the default version of the unmodified loop if we fail the test at runtime, preserving correctness.

# Implementations

- Best fit loop unrolling
  - Based in part on Ma/Carr Paper “Register Pressure Guided Unroll-and-Jam”
    - Utilizes scheduling estimates
    - Estimates Register pressure for each unrolled segment
    - Is not used with Jam nor with Software Pipelining as in the paper
    - Excludes cases based on register pressure estimates
    - Attempts to model code which will produce higher instruction level parallelism
  - Also extended generic size fitting parameters for unrolling
    - Loop overhead now removed from size calculation on unroll segments
  - Used under default parameters only
    - Unroll by 4 or less
    - Size of 128 or less
  - Future work planned

# Implementations

- Register pressure utilities, algorithms and heuristics
  - LRA
    - Utilities used in Unrolling and EBO to guide register pressure decisions
    - Non destructive destination for AVX/FMA4
    - Register pressure based destructive destination
      - High pressure heuristic used to shorten live ranges
      - Decreases register pressure
      - Simplifies complex cases in LRA for AVX/FMA4 code
  - Pre scheduling for register pressure(enabled by flag)
    - Applied generically(level 1)
    - Applied only to unrolled loops(level 2)
    - -CG:pre\_minreg\_level=<1,2>, off is default

# Implementations

- Register pressure utilities, algorithms and heuristics(cont.)
  - EBO
    - Pre scheduling load-execute translation for fp register live ranges gated on register pressure
      - We now load execute under register pressure for fp registers
      - This excludes x87
    - Load-execute translation still un-gated in subsuming reloads to execute instructions after register allocation
    - FMA disassociation
      - Utilize 2 memory homes on un-fused operation in presence of register pressure
      - FMA4 operations only have 1 memory home
      - Generally load-execution on FMA4 under high ipc lowers performance
      - Default is no load execution for FMA4 operations

# Implementations

- CG Loop remainder subsuming to inner loops
  - Discover cases where fully unrolled loops exist in bodies of innermost loops
    - Check if loop body has no flow
    - Extend blocks into a single loop block
  - Benefits
    - Better EBO results(address merging, load-execution, etc)
    - Better Scheduling
    - Better local register allocation
    - Better SIB address generation

# Implementations

- EBO opportunities for Inc/dec
  - Enhanced add/sub for general purpose instruction peephole mapping to include more patterns
    - Optimization is on by default
    - Coverage is 4x on average greater than before
    - Benefits
      - Integer generated code now makes frequent use of the pattern
      - Code emit via assembler is smaller, reducing i-cache miss problems caused by line strides

# Implementations

- **Min/Max for vectorization**

- Added SSE4.2 Packed Vector Min and Max instructions
- Min/Max abstraction existed prior to the addition
  - Optimization is on by default
  - The original code was less optimal
  - Benefits
    - Packed Integer Vector loops now have 1 instruction to cover what was 5
    - Smaller footprint in loops, better latency, higher instruction level parallelism

- **Load Execution for vector operations**

- EBO load execution translation now includes AVX and FMA4 vector operations
- BD allows unaligned load on memory forms of vector operations
- The align penalty on BD is 1/3 of what it was on earlier architectures
- Benefits
  - Allows an answer to register pressure in vector code

# Questions