Software Vector Chaining
M. Anton Ertl
TU Wien

Data Parallelism and SIMD instructions

• Data parallelism in programming problems
• Hardware provides SIMD instructions
  Cray-1 vector instructions, Intel/AMD SSE/AVX, ARM Neon/SVE
  \[
  \text{vmulpd} \ %ymm2, \ %ymm3, \ %ymm1
  \]
• Little programming language support

Programming language support: How?

• Manual Vectorization
• Application vector length
• Opaque, immutable vectors with value semantics
• Vector stack
  \[
  \text{vcomp ( va vb -- vc )}
  \]
  \[
  \text{vdup sf*v vswap vdup sf*v sf+v sfnegatev ;}
  \]

Properties, benefits and drawbacks

• Vectors are immutable (value semantics)
  – Explicit conversion from/to memory
  + gives control to programmer, who can make conversions infrequent
+ Padding to SIMD granularity
+ Aligning to SIMD granularity
+ No aliasing problems
+ Results do not overlap input operands
+ Only explicit dependences
+ Vectors are a separate world
+ Compiler can arrange computations

Implementation

simple sf+v
simple:
\[
\text{vnovaps} \ (\%rdi,\%r10,1),\%ymm0
\]
\[
\text{vaddps} \ (\%rsi,\%r10,1),\%ymm0,\%ymm1
\]
\[
\text{vnovaps} \ %ymm0,\ (\%rds,\%r10,1)
\]
\[
\text{add} \ 0x20,\%r10
\]
\[
\text{cmp} \ %r10,\%r10
\]
\[
\text{ja} \ \text{simple}
\]
fused vcomp
fused:
\[
\text{vnovaps} \ (\%rdi,\%r10,1),\%ymm0
\]
\[
\text{vmulps} \ %ymm0,\%ymm1,\%ymm3
\]
\[
\text{vnovaps} \ %ymm0,\ (\%rds,\%r10,1)
\]
\[
\text{add} \ 0x20,\%r10
\]
\[
\text{cmp} \ %r10,\%r9
\]
\[
\text{ja} \ \text{fused}
\]
... but how?

Who performs vector loop fusion?

Compiler

+ Low run-time overhead
  – High implementation effort?
  – Control-flow may limit fusion
  – Aliasing plays a role

Run-time Library

– High run-time overhead
+ Low implementation effort
+ Fuses across control flow
+ Dependencies resolved

Software Vector Chaining

Implementing a vector operation

Generate code

\[
\text{vdup sf*v vswap vdup sf*v sf+v sfnegatev}
\]

\[
\text{\$241470C0 \ refs= 0 \ bytes=16 \ \$2414760} :14
\]
\[
\text{\$2414810C \ refs= 0 \ bytes=16 \ \$2415150} :15
\]
\[
\text{\$2415030C \ refs= 0 \ bytes=16 \ \$2417300} :19
\]

\[
\text{\$241470C0 \ refs= 0 \ bytes=16 \ \$2414760} :14
\]
\[
\text{\$2414810C \ refs= 0 \ bytes=16 \ \$2415150} :15
\]
\[
\text{\$2415030C \ refs= 0 \ bytes=16 \ \$2417300} :19
\]

\[
\text{\$241470C0 \ refs= 0 \ bytes=16 \ \$2414760} :14
\]
\[
\text{\$2414810C \ refs= 0 \ bytes=16 \ \$2415150} :15
\]
\[
\text{\$2415030C \ refs= 0 \ bytes=16 \ \$2417300} :19
\]
Evaluation

Multiply $50 \times 50$ with $50 \times n$ Double matrix for varying $n$, 500 times on Core i5 6600K (Skylake)

Conclusion

- How to use SIMD instructions for data parallelism?
- Manual vectorization, application vector size, opaque vectors gives freedom to the compiler/library writer
- Software vector chaining
  - Build trace at run-time
  - Compile if not cached
  - Can be implemented as library
  - 315 source lines of code
  - For long vectors $> 2 \times$ as fast as simple
- High per-operation overhead
  - Useful only for long vectors
  - Select between simple and chaining per operation

- [github.com/AntonErtl/vectors](https://github.com/AntonErtl/vectors)
- Paper at ManLang 2018
- [https://www.complang.tuwien.ac.at/papers/ertl18.pdf](https://www.complang.tuwien.ac.at/papers/ertl18.pdf)