Actively analyzing performance
to find microarchitectural bottlenecks and to estimate performance bounds

Kenneth (Kent) Czechowski · Jee Whan Choi (IBM) · Jeff Young · Richard (Rich) Vuduc

July 16, 2015
Workshop on Performance Modeling: Methods and Applications
at International Supercomputing Conference (ISC)
Passive  
(observational)  vs.  
Active  
(experimental)

Many related ideas!

*Environmental modifiers:* DVFS, Gremlins

*Code modifiers:* autotuning, stochastic (super)optimizers
doi: 10.1016/0167-8191(89)90100-2

doi: 10.1145/1498765.1498785
“Desktop GPU” (NVIDIA)

GTX Titan

- 4.0 Tflop/s [16 Gflop/J]
- 240 GB/s [1.3 GB/J]
- 290 W [const=120 W]

“Mobile GPU” (Samsung/ARM)

Arndale GPU

- 33 Gflop/s [8.1 Gflop/J]
- 8.4 GB/s [1.5 GB/J]
- 6.1 W [const=1.3 W]

Efficiency (higher is better)

Intensity (single-precision flop:Byte)

J. Choi et al. (IPDPS’14)
Figure 3a–3c: Roofline model for Intel Xeon, AVO (c) AMD Opteron X4 (Barcelona) (a) Intel Xeon (Clovertown)

“Usual” route: Fix “x” and try to improve “y.”

What about the other direction?
Shiloach-Vishkin algorithm to compute connected components (as labels)

\[
\text{forall } v \in V \text{ do }
\quad \text{label}[v] \leftarrow \text{int}(v)
\]

\[
\text{while } \ldots \text{ do }
\quad \text{forall } v \in V \text{ do }
\quad \quad \text{forall } (u, v) \in E \text{ do }
\quad \quad \quad \text{if } \text{label}[u] < \text{label}[v] \text{ then }
\quad \quad \quad \quad \text{label}[u] \leftarrow \text{label}[v]
\]

Predicted Cycles per instruction [Ivy Bridge]

<table>
<thead>
<tr>
<th>astro-ph</th>
<th>audkw1</th>
<th>auto</th>
<th>coAuthorsDBLP</th>
<th>cond-mat-2003</th>
<th>cond-mat-2005</th>
<th>coPapersDBLP</th>
<th>ecology1</th>
<th>ldoor</th>
<th>power</th>
<th>preferentialAttachment</th>
</tr>
</thead>
</table>

Predicted values:
- Cache.references
- Cache.misses
- Branches
- Mispredictions

Measured

Modeled (counters + lasso regression)
A frontier:

**Performance upper bounds**

Goal of algorithm analysis is to estimate or (lower) bound on $Q$. 

$Q(n; Z) = \# \text{ transfers}$
Lower-bounds on $Q$: Red-blue pebble games
Rule 1: **Input** ("load")

Rule 2: **Output** ("store")

Rule 3: **Compute**

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

Lower-bounds on $Q$: Red-blue pebble games

Inputs (initial)

Outputs
Lower-bounds on $Q$: Red-blue pebble games

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?
Lower-bounds on $Q$: Red-blue pebble games

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?
Lower-bounds on $Q$: Red-blue pebble games

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?
Inputs (initial)

Lower-bounds on $Q$: Red-blue pebble games

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?
Lower-bounds on $Q$: Red-blue pebble games

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?
Lower-bounds on $Q$: Red-blue pebble games

\[ Q(n; Z) = \Omega \left( \frac{n \log n}{\log Z} \right) \]

(Hong & Kung '81)
Lower-bounds on $Q$: Red-blue pebble games

Insight: This representation is computable

Contech: Efficiently Generating Dynamic Task Graphs for Arbitrary Parallel Programs

BRIAN P. RAILING, ERIC R. HEIN, and THOMAS M. CONTE, Georgia Institute of Technology

Fig. 5. Runtime instrumentation design (gray instructions are original code).

Fig. 4. Simple OpenMP program as a Contech task graph.
Kent’s idea:

**Pressure point analysis (PPA)**

Iteratively *rewrite* the input program in a controlled fashion, then *re-analyze* it.

Rewrites need *not* necessarily be semantics preserving!
PPA: Conceptual Example

Tri-Diagonal Elimination
for ( i=1 ; i<n ; i++ ) {
    x[i] = z[i]*( y[i] - x[i-1] );
}

Compute Only

\[
\begin{align*}
\text{vmovsd} & \quad \text{xmm1}, [8+\text{rsi}+\text{r12}] \\
\text{vmovsd} & \quad \text{xmm2}, [16+\text{rsi}+\text{r12}] \\
\text{vsubsd} & \quad \text{xmm0}, \text{xmm1}, \text{xmm0} \\
\text{vmulsd} & \quad \text{xmm3}, \text{xmm0}, [8+\text{rsi}+\text{rbp}] \\
\text{vmovsd} & \quad [8+\text{rsi}+\text{r13}], \text{xmm3} \\
\text{vsubsd} & \quad \text{xmm4}, \text{xmm2}, \text{xmm3} \\
\text{vmulsd} & \quad \text{xmm0}, \text{xmm4}, [16+\text{rsi}+\text{rbp}] \\
\text{vmovsd} & \quad [16+\text{rsi}+\text{r13}], \text{xmm0}
\end{align*}
\]

Memory Access Only

\[
\begin{align*}
\text{vmovsd} & \quad \text{ xmm1}, [8+\text{rsi}+\text{r12}] \\
\text{vmovsd} & \quad \text{ xmm2}, [16+\text{rsi}+\text{r12}] \\
\text{nop} & \quad \\
\text{vsubsd} & \quad \text{ xmm0, xmm1, xmm0} \\
\text{vmulsd} & \quad \text{ xmm3, xmm0, xmm10} \\
\text{nop} & \quad \\
\text{vsubsd} & \quad \text{ xmm4, xmm2, xmm3} \\
\text{vmulsd} & \quad \text{ xmm0, xmm4, xmm12} \\
\text{nop} & \quad \\
\text{vmovsd} & \quad \text{ xmm1, [8+\text{rsi}+\text{r12}] } \\
\text{vmovsd} & \quad \text{ xmm2, [16+\text{rsi}+\text{r12}] } \\
\text{nop} & \quad \\
\text{vmovsd} & \quad \text{ xmm3, [8+\text{rsi}+\text{rbp}] } \\
\text{vmovsd} & \quad \text{ [8+\text{rsi}+\text{r13}], xmm3 } \\
\text{nop} & \quad \\
\text{vmovsd} & \quad \text{ xmm0, [16+\text{rsi}+\text{rbp}] } \\
\text{vmovsd} & \quad \text{ [16+\text{rsi}+\text{r13}], xmm0 }
\end{align*}
\]

Perturbations do not need to preserve the semantic meaning
Concrete Example: L1D Bank Conflicts

- Bank 0: 0-7
- Bank 1: 8-15
- Bank 2: 16-23
- Bank 3: 24-31
- Bank 4: 32-39
- Bank 5: 40-47
- Bank 6: 48-55
- Bank 7: 56-63

64 Byte Entries

movpd xmm1, [r12 + 16]
movpd xmm2, [r12 + 112]
Concrete Example: L1D Bank Conflicts

movpd xmm1, [r12 + 16]

movpd xmm2, [r12 + 88]
**Concrete Example: L1D Bank Conflicts**

```
[ 8 +rsi+r12] -> [X+rsi+r12]
[ 8 +rsi+r14] -> [X+rsi+r14]
[ 8 +rsi+rbp] -> [X+rsi+rbp]
[ 8 +rsi+r13] -> [X+rsi+r13]
[16+rsi+rbp] -> [X+rsi+rbp]
[16+rsi+r13] -> [X+rsi+r13]
```

*Assume rsi, r12, r13, r14, and rbp are 64-byte aligned*

**Original**
- `vmovsd xmm1, [8+rsi+r12]`
- `vmovsd xmm2, [8+rsi+r14]`
- `vsubsd xmm0, xmm1, xmm0`
- `vmulsd xmm3, xmm0, [8+rsi+rbp]`
- `vmovsd [8+rsi+r13], xmm3`
- `vsubsd xmm4, xmm2, xmm3`
- `vmulsd xmm0, xmm4, [16+rsi+rbp]`
- `vmovsd [16+rsi+r13], xmm0`

**Perturbed Version**
- `vmovsd xmm1, [8+rsi+r12]`
- `vmovsd xmm2, [16+rsi+r14]`
- `vsubsd xmm0, xmm1, xmm0`
- `vmulsd xmm3, xmm0, [8+rsi+rbp]`
- `vmovsd [8+rsi+r13], xmm3`
- `vsubsd xmm4, xmm2, xmm3`
- `vmulsd xmm0, xmm4, [16+rsi+rbp]`
- `vmovsd [16+rsi+r13], xmm0`
IDENTIFYING OOO-DEFICIENCIES

Original

Cycles per Iteration: 31.51 cycles

Scrambled

Cycles per Iteration: 19.65 cycles
Our Vision for Performance Analysis

Can we account for all lost cycles?

```plaintext
for ( k=0 ; k<n ; k++ ) {
    x[k] = u[k] + r*( z[k] + r*y[k] ) + 
    t*( u[k+3] + r*( u[k+2] + r*u[k+1] ) ) + 
    t*( u[k+6] + r*( u[k+5] + r*u[k+4] ) ) ;
}
```

Automated battery of experiments

- Frontend bottlenecks
- Scheduling resource conflicts
- Data bypass delays
- Cache latency stalls
- Memory disambiguation conflicts
- Retirement bandwidth
CONCLUSION / SUMMARY

Major Contribution: Active Performance Analysis
Status: Proof of concept
Gaps:
- Comprehensive set of experiments
- Scale beyond the core
- Generalize to additional microarchitectures

Cross-Pollination:
- Software optimization
- Autotuning and super-optimizing compilers
- Hardware-software codesign