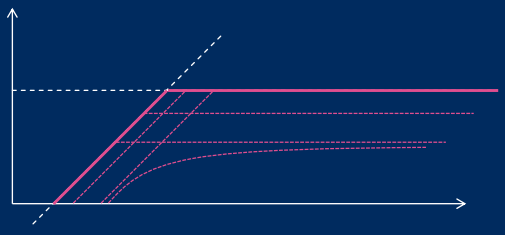


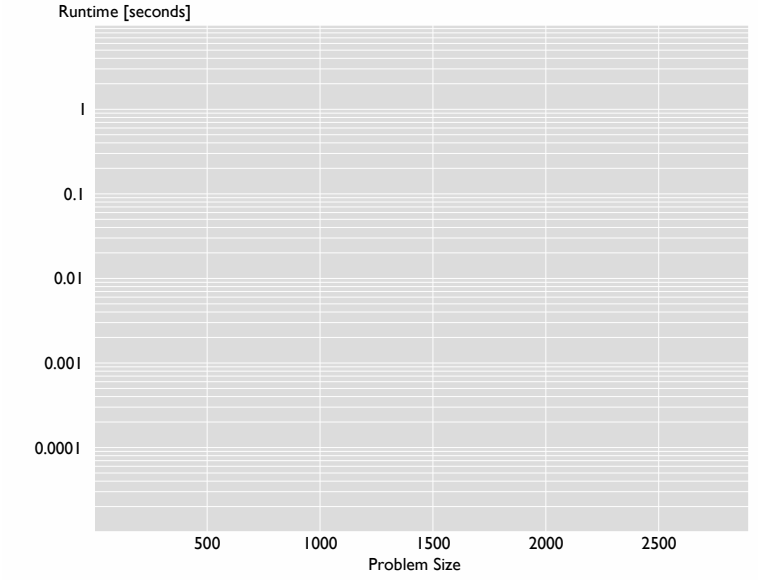
Extending the Roofline Model: Bottleneck Analysis with Microarchitectural Constraints

Victoria Caparrós Cabezas
Markus Püschel

Part of PhD work

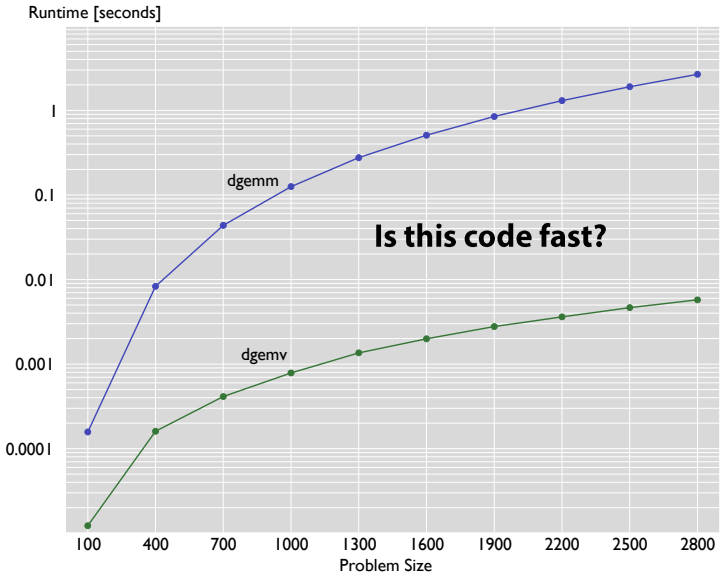
Computer Science
ETH zürich

Measuring Runtime



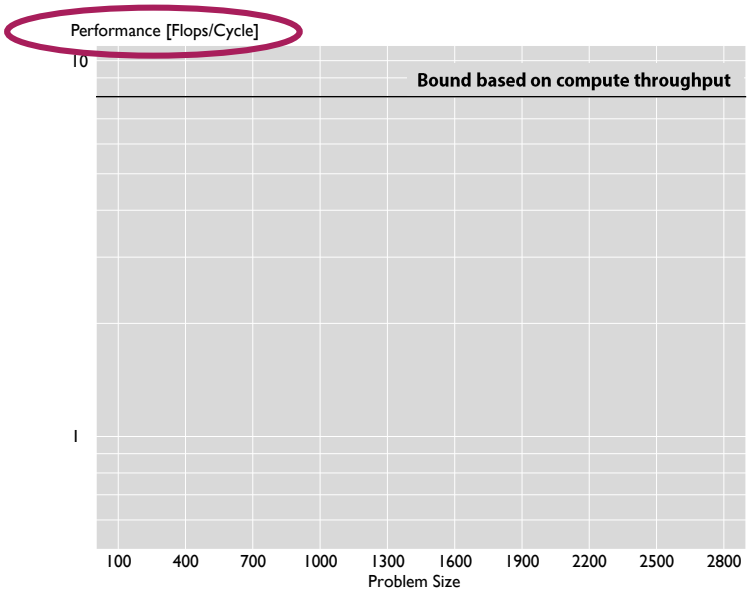
2

Measuring Runtime



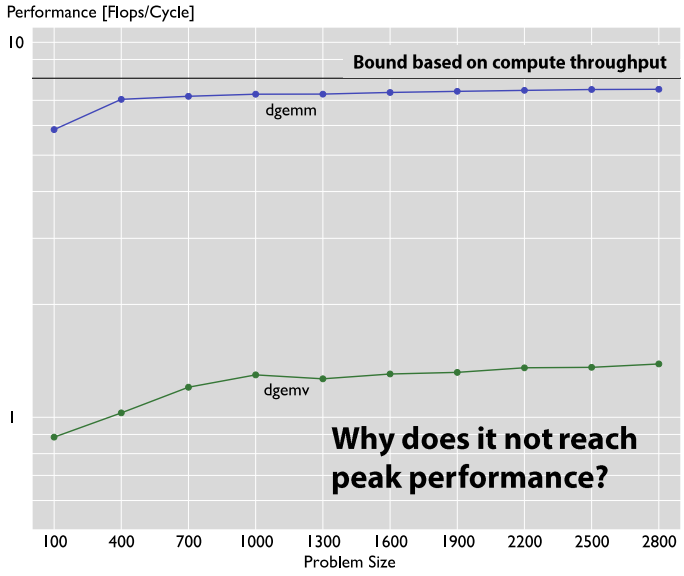
3

Measuring Performance



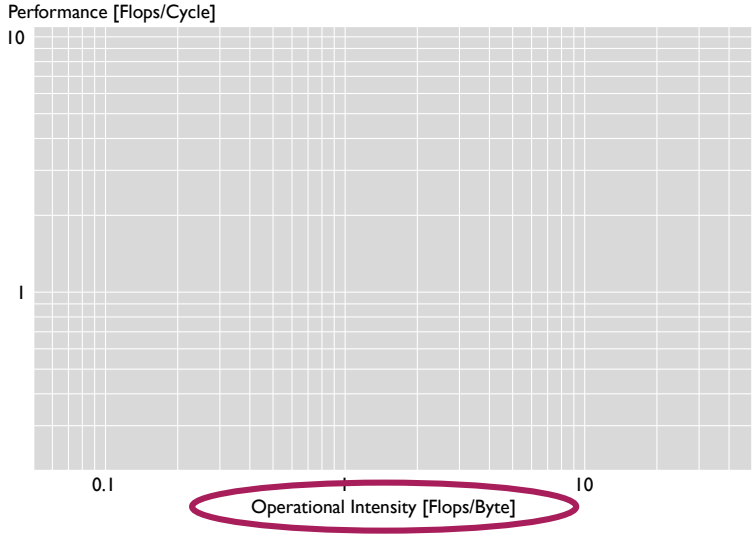
4

Measuring Performance



5

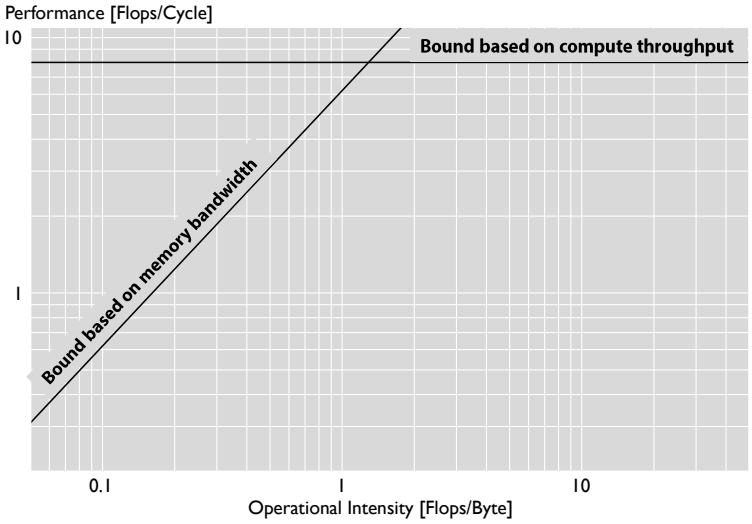
Roofline Plot



Williams et al. "Roofline: An Insightful Visual Performance Model for Multicore", Communications of the ACM, 2009

6

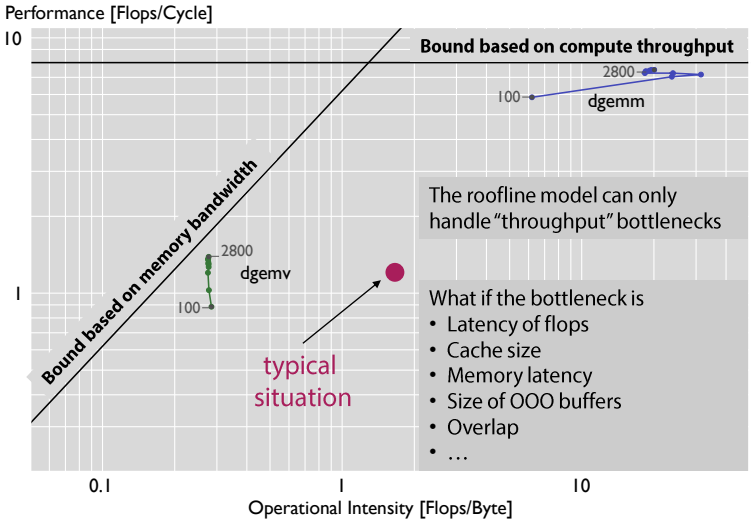
Roofline Plot



Williams et al. "Roofline: An Insightful Visual Performance Model for Multicore", Communications of the ACM, 2009

7

Roofline Plot

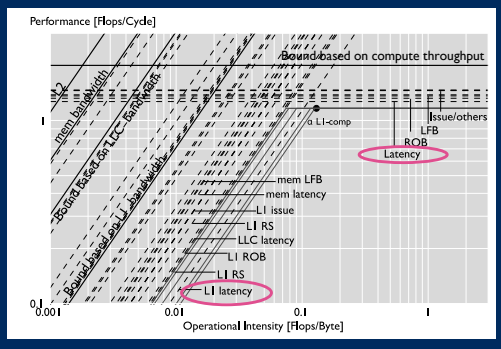


Measurements: Ofenbeck et al. "Applying the Roofline Model" Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, pp. 76-85

8

Goal

Extended Roofline Model (ERM) to include additional hardware-related bottlenecks



Performance bottlenecks:
1. Latency of flops
2. Latency of L1 accesses

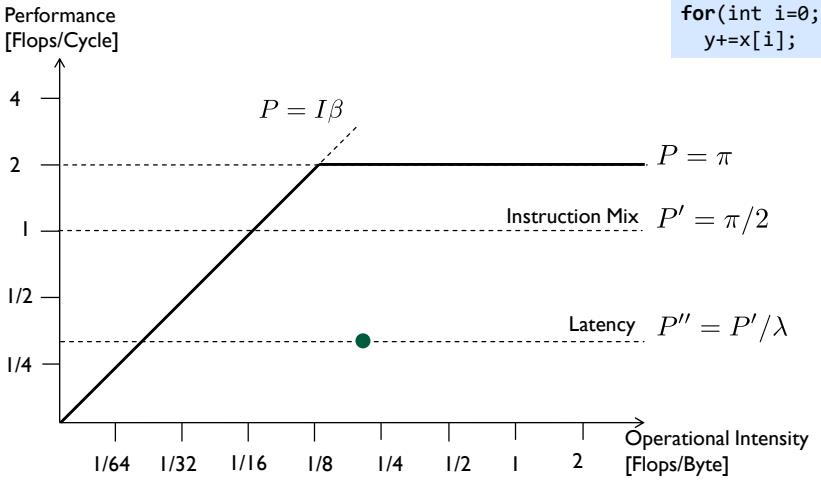
Current limitation:
single core (with SIMD) only

Applications

- Code optimization
- Assess the impact of hardware upgrades

Example Challenge: Latency

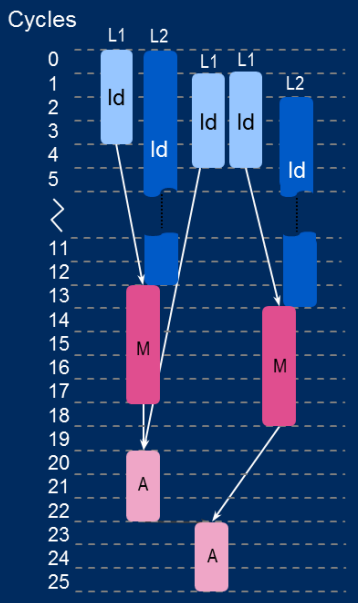
```
int n = 1000;
x=(float*)malloc(…);

//Vector sum reduction
for(int i=0; i<n; i++)
  y+=x[i];
```



Challenge: What if the latency only affects some of the operations?

Our solution:

Detailed breakdown of the computation time through a DAG-based performance model



Input: C Code plus input

```
int main(){

int n = 1000;
A=(double*)_mm_malloc(…);
…

for(int i=0; i<n; i++)
 for(int j=0; j<n; j++)
  for(int k=0; k<n; k++)
    C[i][j]+=A[i][k]*B[k][j];

}
```

1 `clang -emit-llvm -c -O3 …`

LLVM IR
(no registers, SSA, some vector instructions)

```
%5 = trunc i64 %indvars.iv87 to i32
%idxprom13.us.us.unr = sext i32 %add12 to i64
%arrayidx14.us.us.unr = getelementptr %B,
%6 = load double* %arrayidx14.us.us.unr, align 8,
%mul15.us.us.unr = fmul double %3, %6
…
```

2 **LLVM dynamic trace generator from Railing et al. [TACO 2015]**

Dynamic execution trace (DAG)

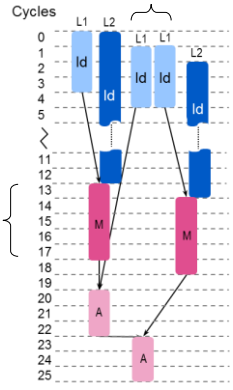
```
%9 = load double* %arrayidx.us, align 8, !dbg !389, !tbaa !384 (0x3dc8a48)
%6 = load double* %arrayidx17.us, align 8, !dbg !385, !tbaa !384 (0x3dc7818)
%7 = load double* %arrayidx18.us, align 8, !dbg !385, !tbaa !384 (0x3dc7aa8)
%mul.us = fmul double %6, %7, !dbg !385 (0x3dc7d10)
…
```

Input: Microarchitectural Parameters

Microarchitectural Parameters	Unit	Value
Throughput		
π_A, π_M	flops/cycle	1,1
β_{mem}, β_{L_i}	doubles/cycle	1, 2/1, 4, 4
ϕ	instr./cycle	4
Latency		
λ_A, λ_M	cycles	3, 5
μ_{mem}, μ_{L_i}	cycles	100, 4, 12, 30
Capacity		
γ_{L_i}	bytes	32K, 256K, 20M
φ	bytes	64
γ_{buf}	# slots	168, 54, 36, 64, 10

3 **Tomasulo's scheduling algorithm**
ERM (Our Tool)

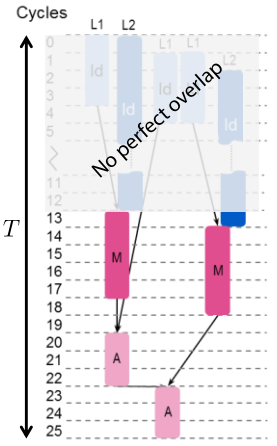
Output: Scheduled DAG

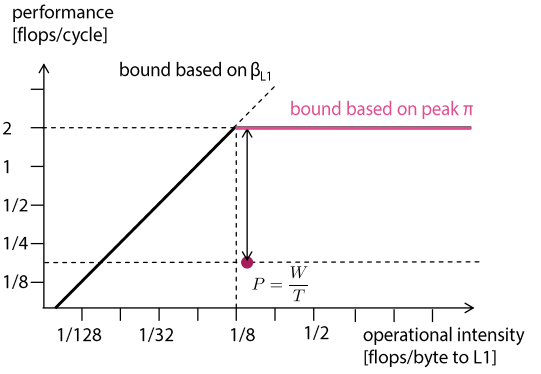


+ data from its analysis

12

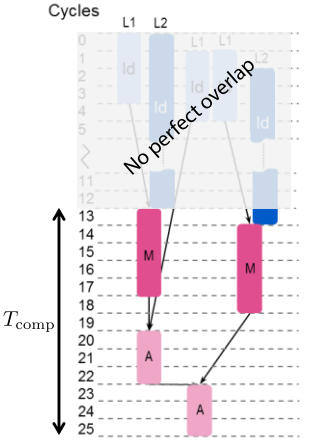
Utilization-Based Bottleneck Modeling

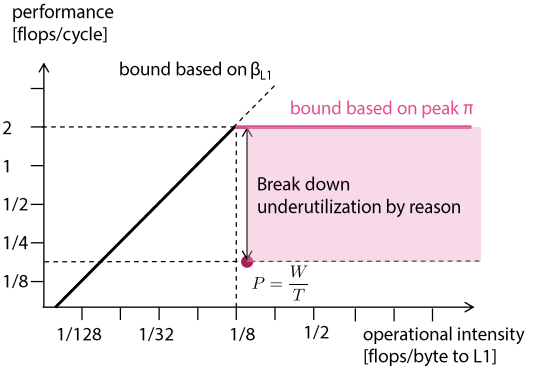




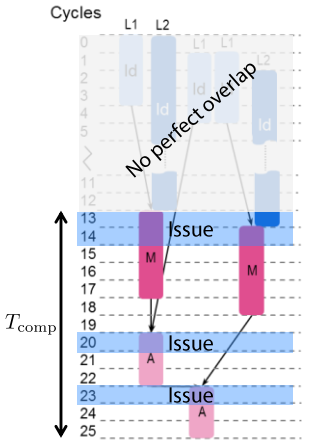
Where are execution cycles spent so that peak performance is not reached?

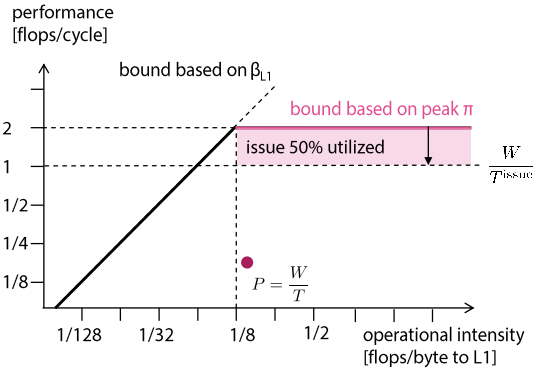
Utilization-Based Bottleneck Modeling





Utilization-Based Bottleneck Modeling

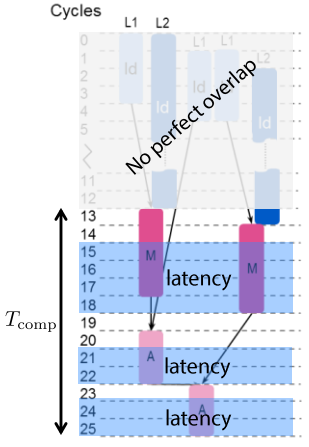


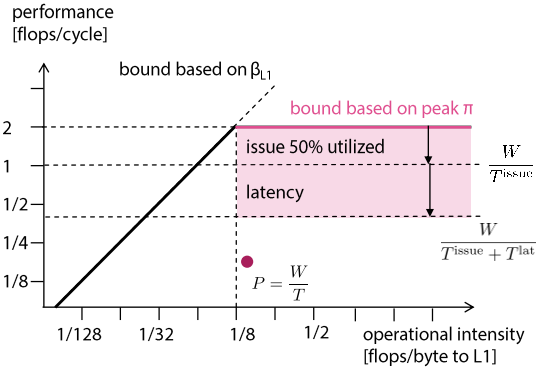


T^{issue} = Number of cycles in which nodes are issued $\quad T^{\text{issue}} = 4$

15

Utilization-Based Bottleneck Modeling





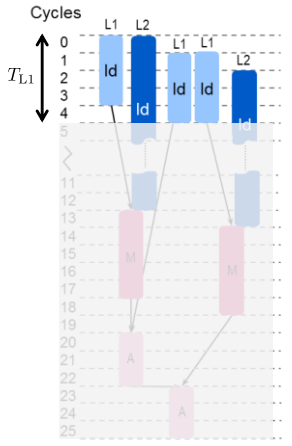
T^{issue} = Number of cycles in which nodes are issued $\quad T^{\text{issue}} = 4$

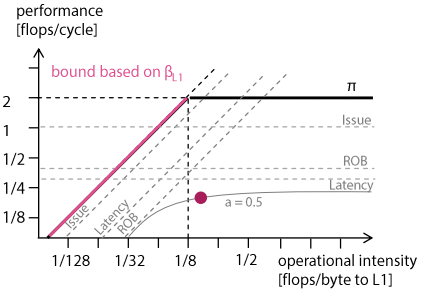
T^{lat} = Number of cycles in which nodes are executed, but not issued $\quad T^{\text{lat}} = 8$

16

Utilization-based Bottleneck Modeling

Similar procedure for each level of the memory hierarchy





Finally, compute overlap (with compute) lines

Merging Roofline Plots for Memory Hierarchy

Original roofline model:

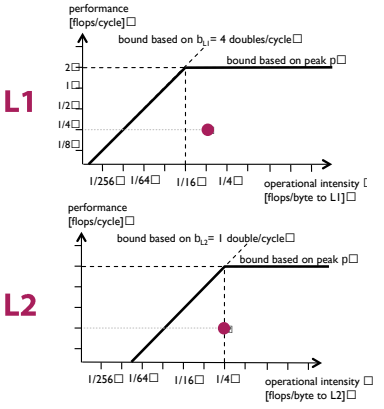
$$I_x = \frac{W}{Q_x}$$

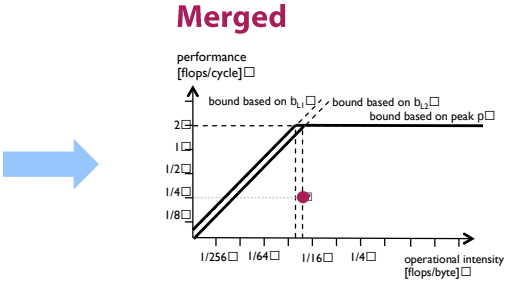
$$P_x \leq \min(I_x \beta_x, \pi)$$

Redefinition of operational intensity:

$$I = \frac{W}{Q} = \frac{W}{Q_{L1} + Q_{L2} + Q_{L3} + Q_{mem}} \qquad I_x = I\frac{Q}{Q_x}$$

$$P_x \leq \min(I_x \beta_x, \pi) = \min(I\frac{Q}{Q_x}\beta_x, \pi)$$

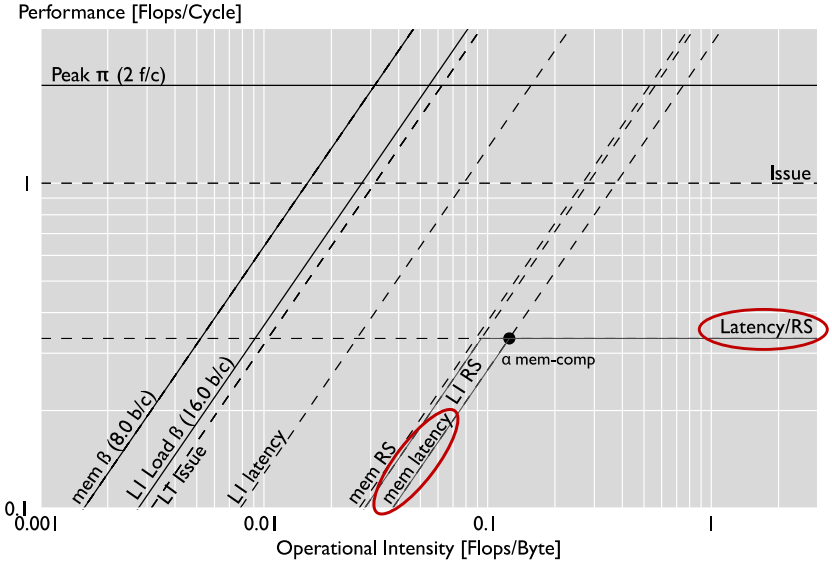




Some Results

19

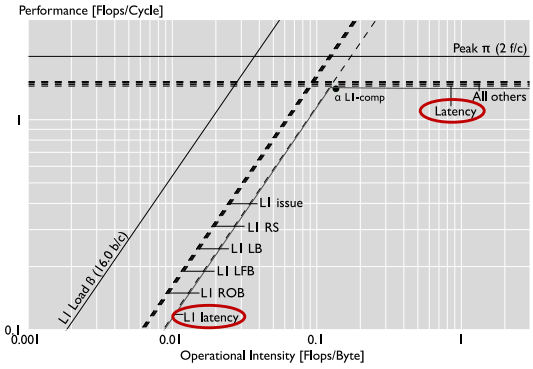
Vector Sum Reduction (cold cache, n = 5 x 10^6)



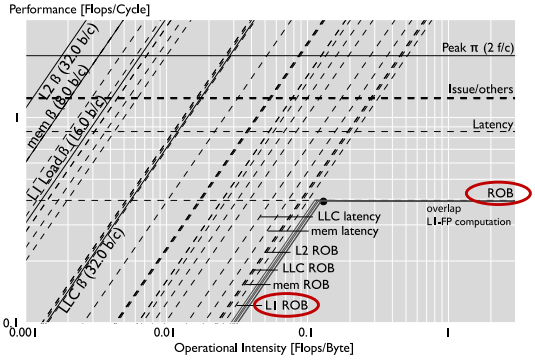
20

FFT NR (warm cache)— Increasing Size

Size 2^{10}



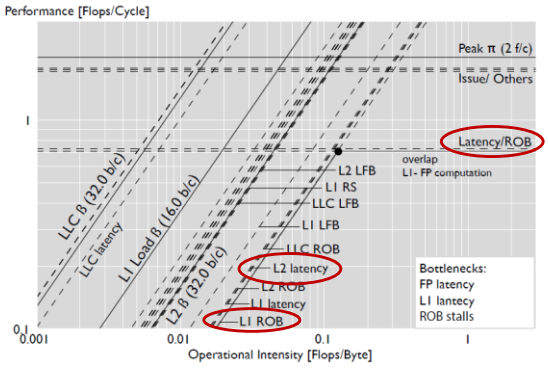
Size 2^{20}



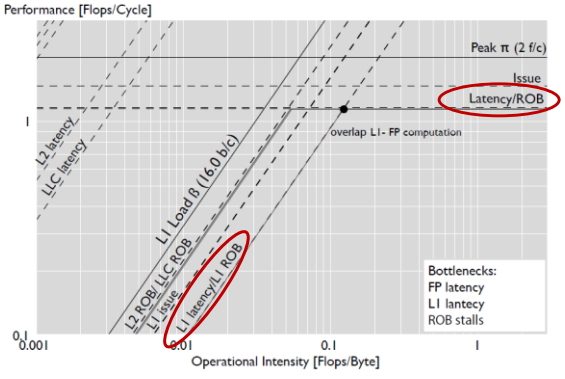
21

MMM Warm Cache: Triple Loop vs. Blocked (n = 500)

Triple loop



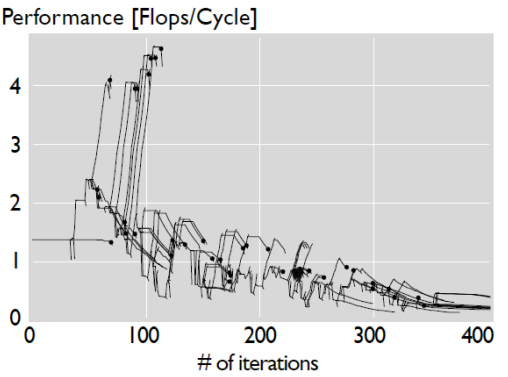
Blocked for L1 (b = 50)



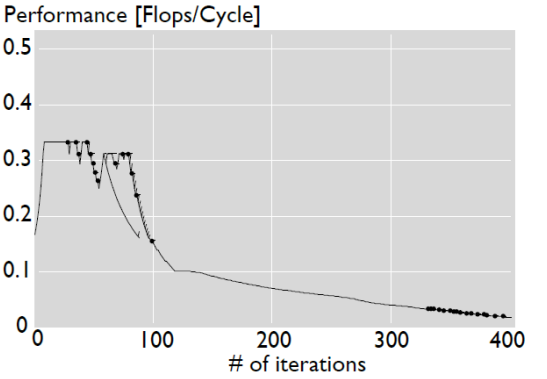
22

Followup Work — Balancing Processor for Kernel

Hill climbing algorithm to find balanced processors (= good match SW/HW)



(a) Livermore loop 1 (small), initial configuration Sandy Bridge.



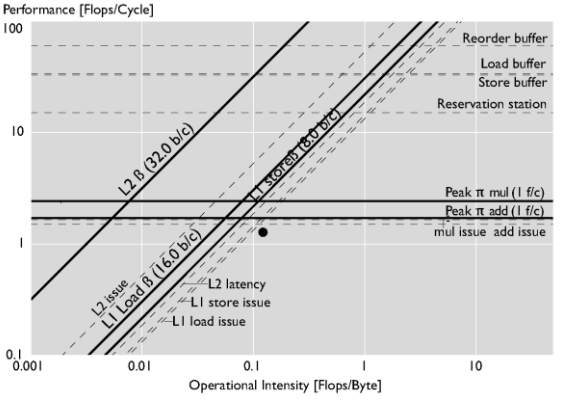
(b) Livermore loop 1 (small), initial configuration Cortex-A9.

LL 1: Hydrodynamics computation fragment

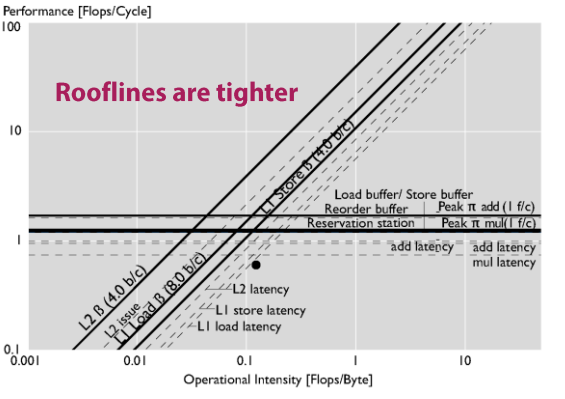
23

Effect on Roofline Plot (Example)

On Sandybridge

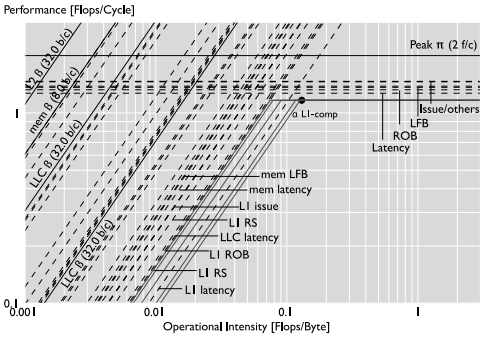


On a balanced processor



24

Conclusions



Extended roofline model for computational kernels

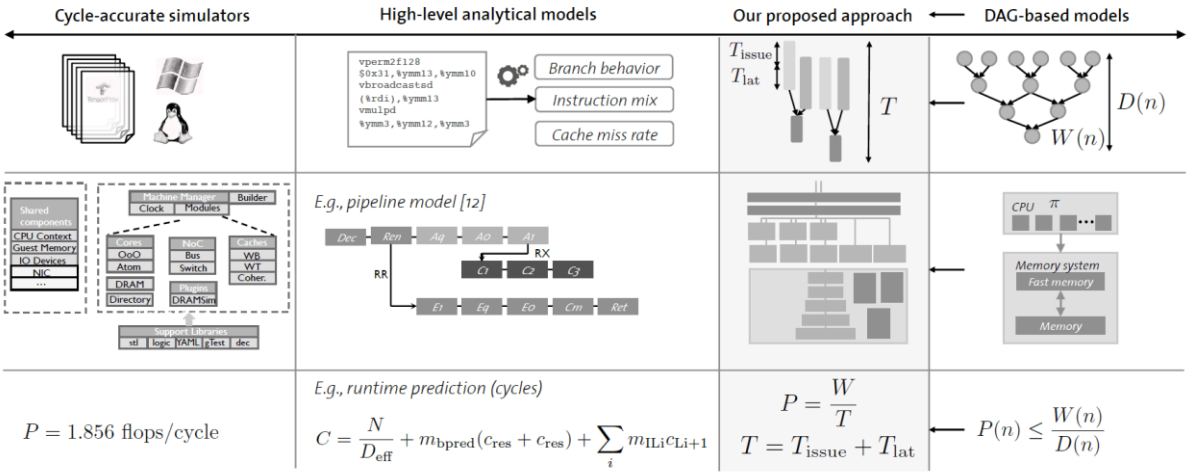
- Memory hierarchy
- Throughput and latency
- Size of caches and OOO buffers
- **To date: Single core with SIMD**

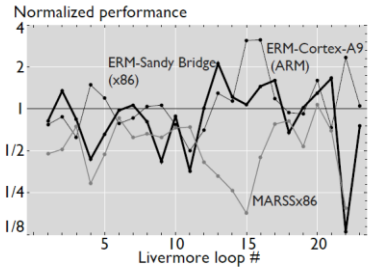
Code: DAG from LLVM interpreter
Processor: 30+ parameters

Basic Idea: Schedule and analyze DAG

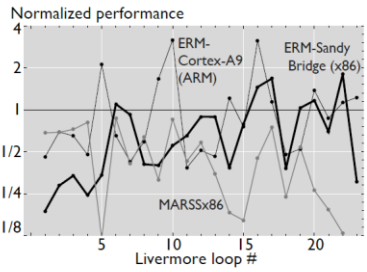
Tool ERM:
https://github.com/caparrov/llvm-performance

Paper:
Victoria Caparrós Cabezas and Markus Püschel **Extending the Roofline Model: Bottleneck Analysis with Microarchitectural Constraints** Proc. IISWC, pp. 222-231, 2014

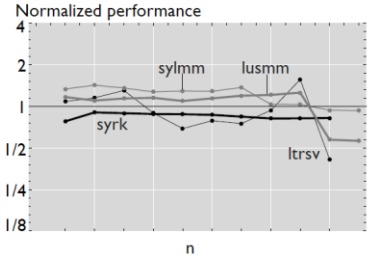




(a) Livermore loops small sizes (scalar code).



(b) Livermore loops large sizes (scalar code).



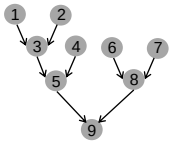
(c) Linear algebra subroutines (vector code). For syrk and ltrsv, $n = 16 : 16 : 160$, for lusmm and sylmm $n = 16 : 8 : 96$ (Matlab notation).

ERM: DAG-Based Performance Model

Application

```
int main(){

int n = 1000;
A=(double*)_mm_malloc(…);
…
for(int i=0; i<n; i++)
 for(int j=0; j<n; j++)
  for(int k=0; k<n; k++)
    C[i][j]+=A[i][k]*B[k][j];

}
```

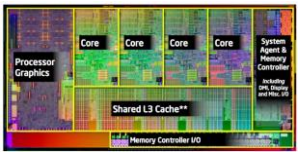
Model



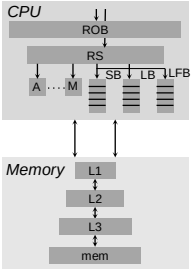
Parameters (6)

Node type (x)	N # nodes
Computation	W
A	W_{A}
M	W_{M}
Memory	Q
L1	Q_{L1}
L2	Q_{L2}
L3	Q_{L3}
mem	Q_{mem}

Processor



Model



Parameters (22)

Microarchitectural Parameters	Unit
Throughput	
π_{A}, π_{M}	flops/cycle
β_{mem}, β_{L_i}	doubles/cycle
ϕ	instr./cycle
Latency	
λ_{A}, λ_{M}	cycles
μ_{mem}, μ_{L_i}	cycles
Capacity	
γ_{L_i}	bytes
φ	bytes
γ_{buf}	# slots

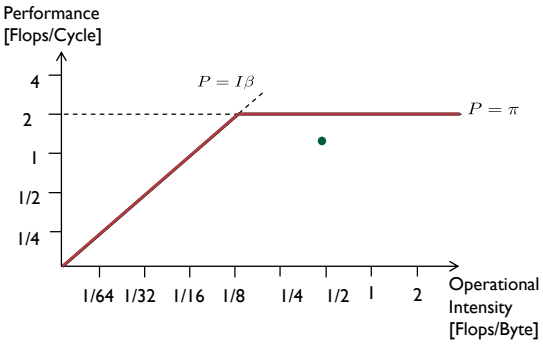
Simple Example: Overlap

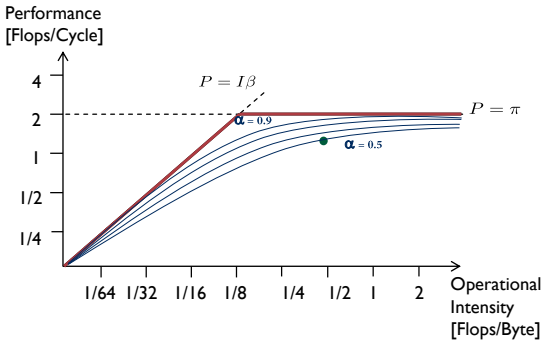
Perfect overlap

$$T = \max(T_{comp}, T_{mem})$$
$$P = \min(\beta I, \pi)$$

Overlap 0 ≤ α ≤ 1

$$T = T_{comp} + T_{mem} - \alpha \min(T_{comp}, T_{mem})$$
$$P = \frac{I}{\frac{I}{\pi} + \frac{1}{\beta} - \alpha \min(\frac{I}{\pi}, \frac{1}{\beta})}$$





Challenge 2: How to get overlap information?