April 22, 2025

Announcements

Goals

Review

$P \leq \min(P_{peak}, I \cdot b)$

req/s

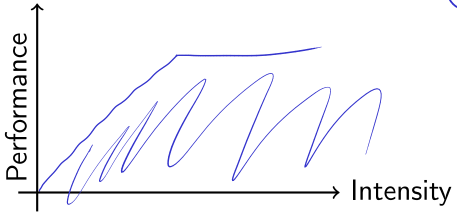
A Graphical Representation: 'Roofline'

Plot (often log-log, but not necessarily):

- X-Axis: Intensity
- Y-Axis: Performance

What does our inequality correspond to graphically?

$$P \leq \min(P_{\text{peak}}, I \cdot b)$$



What does the shaded area mean?

Theoretically attainable perf.

Hager et al. '17

Refining the Model

- P_{max}: Applicable peak performance of a loop, assuming that data comes from the fastest data path (this is not necessarily P_{peak})
- Computational intensity ("work" per byte transferred) over the slowest data path utilized
- b: Applicable peak bandwidth of the slowest data path utilized

Hager et al. '17

Practical Tool: `llvm-mca`

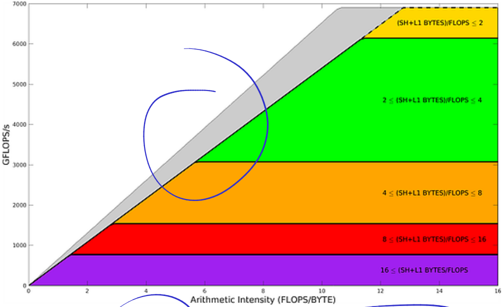
Question: Where to obtain an estimate of $P_{\max}$?

Demo: perf/Forming Architectural Performance Expectations

What does llvm-mca do about memory access? / the memory hierarchy?

assume $\nexists$

An Example: Exploring Titan V Limits



- Memory bandwidth: 652 GB/s theoretical, 540 GB/s achievable
- Scratchpad / L1 throughput:
  80 (cores) x 32 (simd width) x 4 (word bytes) x 1.2 (base clock) ~=
  12.288 TB/s
- Theoretical peak flops of 6.9 TFLOPS/s [Wikipedia]

Warburton '18

Rooflines: Assumptions

What assumptions are built into the roofline model?

- Throughput-only
- Perfect overlap
- Only dominant bottleneck

Important to remember:

- It is what you make of it–the better your calibration, the more info you get
- But: Calibrating on experimental data loses predictive power (e.g. SPMV)

Outline

Timing Experiments: Pitfalls

What are potential issues in timing experiments? (What can you do about them?)

- Warm-up effects
- Noise
  - timer Δt
  - clock type
  - RTC, PIT, APIC, TSC, ...
  - Overheads

Timing Experiments: Pitfalls (part 2)

What are potential issues in timing experiments? (What can you do about them?)

- NUMA placement numactl.
- Thread migration
- Uninitialized memory (or calloc) Bw/thread
- Denormals
- Frequency scaling
- Other users
- Hyperthreading.
- Caches/TLBs

Outline

Profiling: Basic Approaches

Measurement of "quantities" relating to performance

- Exact: Through binary instrumentation (valgrind/Intel Pin/. . .)
- Sampling: At *some* interval, examine the program state

We will focus on profiling by *sampling*.

Big questions:

- What to measure?
- At what intervals?

Defining Intervals: Performance Counters

A *performance counter* is a counter that increments every time a given event occurs.

What events?

- Demo: perf/Using Performance Counters
- see also Intel SDM, Volume 3

Interaction with performance counters:

- Read repeatedly from user code
- Interrupt program execution when a threshold is reached
- Limited resource!
  - Only a few available: 4-8 per core
  - Each can be configured to count one type of event
  - Idea: Alternate counter programming at some rate (requires steady-state execution!)

Profiling: What to Measure

- Raw counts are hard to interpret
- Often much more helpful to look at *ratios* of counts per core/subroutine/loop/...

What ratios should one look at?
Demo: perf/Using Performance Counters

Profiling: Useful Ratios

Basic examples:

- (Events in Routine 1)/(Events in Routine 2)
- (Events in Line 1)/(Events in Line 2)
- (Count of Event 1 in X)/(Count of Event 2 in X)

Architectural examples:

IPC

L1 miss / L1 access

stalled-cycles-frontend / cycles

backend

Issue with 'instructions' as a metric?

uOps vs. instructions.

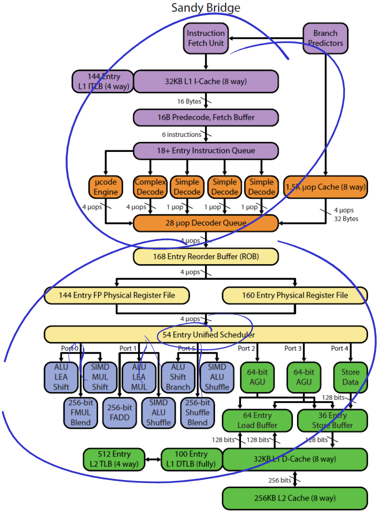
“Top-Down” Performance Analysis

Idea: Account for useful work per available issue slot
What is an issue slot?

a cycle at a port

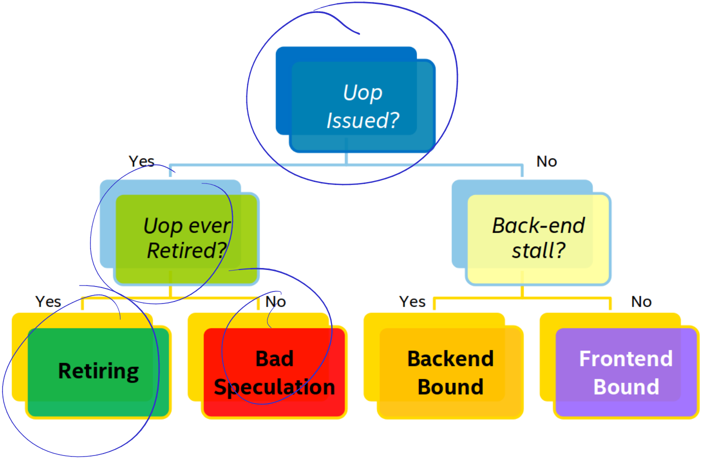
[Yasin ‘14]

Issue Slots: Recap



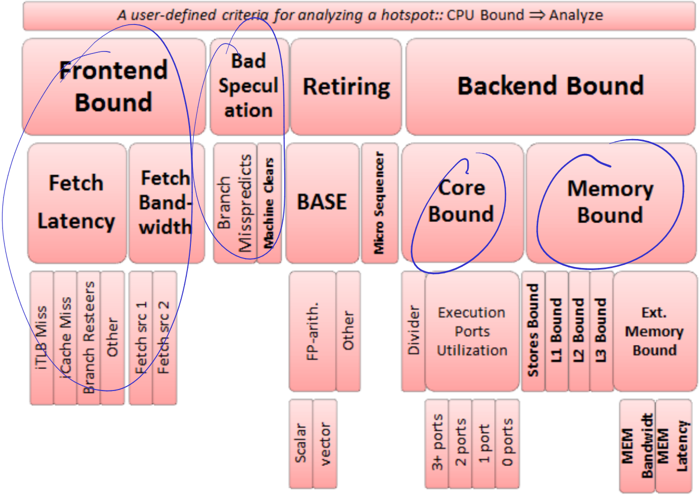
[David Kanter / Realworldtech.com]

What can happen to an issue slot: at a high level?



[Yasin '14]

What can happen to an issue slot: in detail?



[Yasin '14]

Outline

Demo: Performance Counters

Show the rest of:
Demo: perf/Using Performance Counters

Outline

Reduction

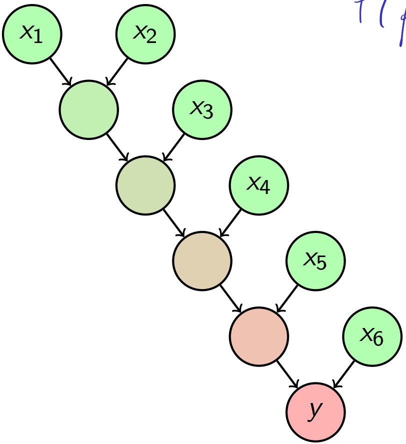
$$\sum_{i=1} a_i$$

$$y = f(\cdots f(f(x_1, x_2), x_3), \ldots, x_N)$$

where N is the input size.
Also known as

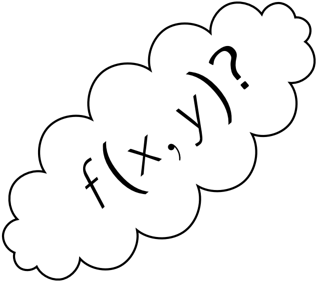
- Lisp/Python function `reduce` (Scheme: `fold`)
- C++ STL `std::accumulate`

Reduction: Graph



$f(f(x,y),z)$
$= f(x, f(y,z))$

Approach to Reduction



Can we do better?

"Tree" very imbalanced. What property of f would allow 'rebalancing'?

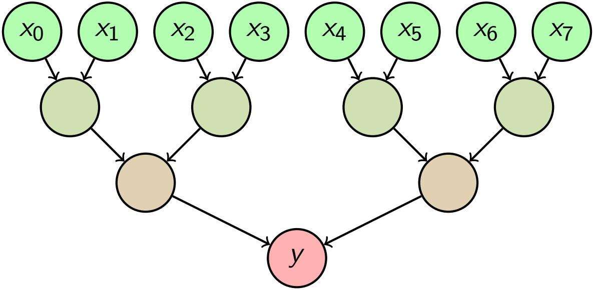
$$f(f(x, y), z) = f(x, f(y, z))$$

Looks less improbable if we let $x \circ y = f(x, y)$:

$$x \circ (y \circ z)) = (x \circ y) \circ z$$

Has a very familiar name: *Associativity*

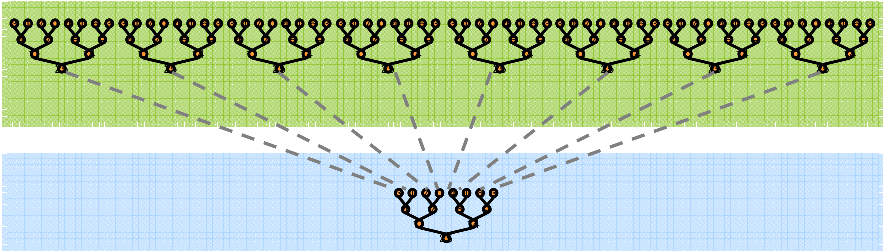
Reduction: A Better Graph



Processor allocation?

Mapping Reduction to SIMD/GPU

- Obvious: Want to use tree-based approach.
- Problem: Two scales, Work group and Grid
  - to occupy both to make good use of the machine.
- In particular, need synchronization after each tree stage.
- Solution: Use a two-scale algorithm.



In particular: Use multiple grid invocations to achieve inter-workgroup synchronization.

Map-Reduce

But no. Not even close.

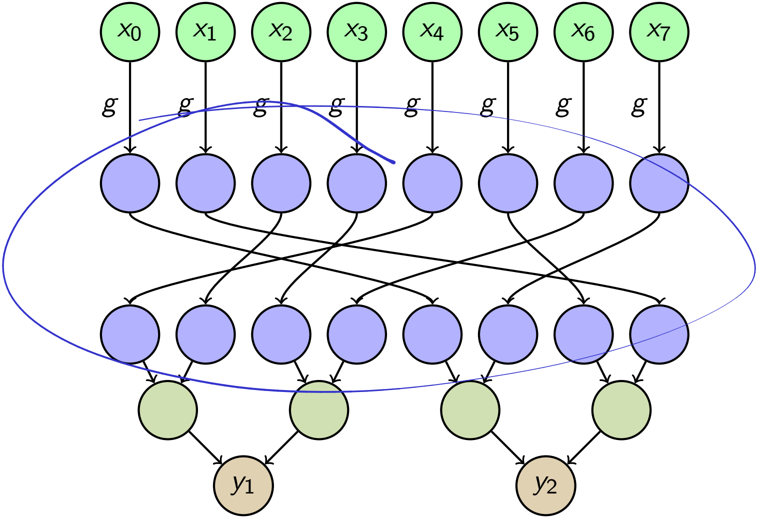
Sounds like this:

$$y = f(\cdots f(f(g(x_1), g(x_2)),\\ g(x_3)), \ldots, g(x_N))$$

where N is the input size.

- Lisp naming, again
- Mild generalization of reduction

Map-Reduce: Graph



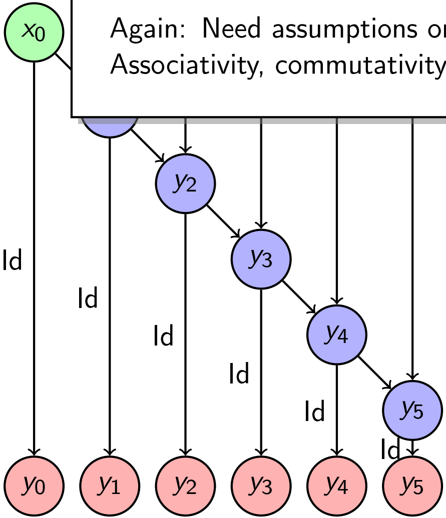
Scan

$$
\begin{aligned}
y_1 &= x_1 \\
y_2 &= f(y_1, x_2) \\
\vdots &= \vdots \\
y_N &= f(y_{N-1}, x_N)
\end{aligned}
$$

where N is the input size. (Think: N large, $f(x, y) = x + y$)

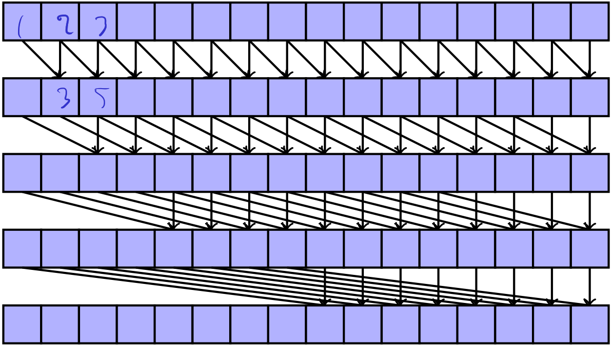
- Prefix Sum/Cumulative Sum
- Abstract view of: loop-carried dependence
- Also possible: Segmented Scan

Scan: Graph



Again: Need assumptions on f.
Associativity, commutativity.

Scan: Implementation



N log N

Work-efficient?

Scan: Implementation II

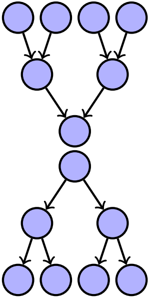
Two sweeps: Upward, downward, both tree-shape

On upward sweep:

- Get values `L` and `R` from left and right child
- Save L in local variable `Mine`
- Compute `Tmp = L + R` and pass to parent

On downward sweep:

- Get value `Tmp` from parent
- Send `Tmp` to left child
- Sent `Tmp+Mine` to right child



Scan: Examples

Name examples of Prefix Sums/Scans:

Sort

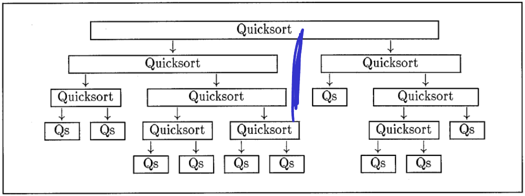
Filter

Segmentation

Data-parallel language: Goals

Goal: Design a full data-parallel programming language

Example: What should the (asymptotic) execution time for Quicksort be?



Question: What parallel primitive could be used to realize this?

NESL Example: String Search

```
teststr = "string  strap asop string" : [char]
>>> candidates = [0:#teststr-5];
candidates = [0, 1, 2, 3, .... : [int]
>>> {a == `s: a in teststr -> candidates};
it = [T, F, F, F, F, F, F, T, F, F....] : [bool]
>>> candidates = {c in candidates;
...         a in teststr -> candidates | a == `s};
candidates = [0, 7, 13, 20, 24] : [int]
>>> candidates = {c in candidates;
...         a in teststr -> {candidates+1:candidates}
...         | a == `t};
```

- Work and depth of this example?
- NESL specifies work and depth for its constructs
- How can scans be used to realize this?

Blelloch '95

Outline

Outline

Basic Object: Presburger Set

Think of the problem statement here as representing an arbitrary-size (e.g.: dependency) graph.
Presburger sets correspond to a subset of predicate logic acting on tuples of integers.
Important: Think of this as a mathematical tool that can be used in many settings.

$\{ (i,j,k) \} \subset \mathbb{Z}^3$

$\{ n \to (i,j,k) \} \subset \mathbb{Z}^4$

$\{ n \quad (i,j,k) \to (i',j',k') : \quad \} \subset \mathbb{Z}^7$

Basic Object: Presburger Set

$\mathbb{Z}^2$

Terms:

- Variables, Integer Constants
- $+, -$
- $\lfloor \cdot / d \rfloor$

$i \cdot 5$

i/j

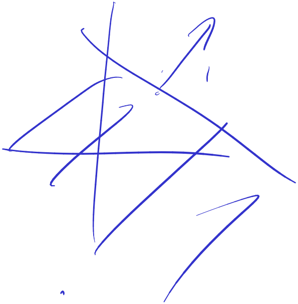
i/n

$5 \cdot i + 3j \le 10$

$\begin{pmatrix} 5 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} \le 10$

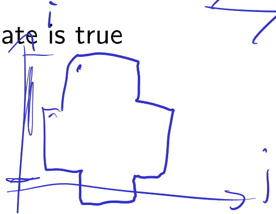
Predicates:

- (Term) $\le$ (Term)
- (Pred) $\wedge$ (Pred), (Pred) $\vee$ (Pred), $\neg$(Pred)
- $\exists v$: (Pred)(v)



Sets: integer tuples for which a predicate is true

Verdoolaege '13



Presburger Sets: Reasoning

What's "missing"? Why?

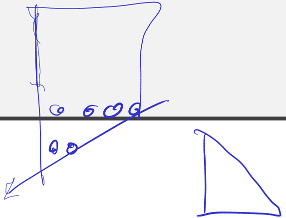
Why is this called 'quasi-affine'?

$\{(i) : 0 \le i < 10 \land \exists j : i = 2j\}$

Presburger Sets: Reasoning

What do the resulting sets have to do with polyhedra? When are they convex?

Why polyhedra? Why not just rectangles?



Demo: Constructing and Operating on Presburger Sets

Demo: lang/Operating on Presburger Sets

Making Use of Presburger Sets

- Loop Domains
- Array Access Relations (e.g. write, read: per statement)
- Schedules, with "lexicographic time"
- Dependency graphs
- (E.g. cache) interference graphs

Q: Specify domain and range for the relations above.

Example: Dependency Graph

Given:

- Write access relation W: Loop domain $\to$ array indices
- Read access relation R
- Schedule S for statement S_i: Loop domain $D \to$ lex. time of statement instance
- Relation $\prec$: Lexicographic 'before'

Find the dependency graph:

Verdoolaege '13