Viewing Application/Machine Interactions through Idioms

M. Laurenzano, L. Carrington, A. Jundt, A. Tiwari, J. Peraza, **W. Ward**, R. Campbell

{michaell, laura.carrington, adam.jundt, ananta.tiwari, joshua.peraza}@epanalytics.com
{william.ward, roy.campbell}@hpc.mil

Why Energy/Performance Models?

- System designers
  - how does my app perform on different platforms?
  - what are the tradeoffs between platforms?
  - what is the right (mix of) platforms for my workload?
- Application developers
  - which platform will benefit my app? by how much?
  - how much effort will it take?
  - where are the performance leaks?
- Users
  - which platform will best utilize my allocation?

The Challenges of Exascale

- Energy efficiency is a major constraint
  - A solution - specialized platform/accelerators (e.g., GPGPU, FPGA, MIC, DSP)
- Models are needed to rapidly evaluate diverse HW options
  - Our understanding of applications needs to span diverse hardware options
  - Example: GPGPU vs. traditional processor
    - memory behavior
    - communications
- Key idea: treat applications as combinations of idioms

What is an Idiom?

- Definition - an ***idiom*** is a small computational or memory access pattern
- Some clarification
  - An idiom can be expressed in a few lines of code
  - idiom != dwarf
    - A dwarf is an algorithm/description of a problem
- Examples
  - Gather `A[i] = B[C[i]]`
  - Reduction `s += A[i]`
  - Stream `A[i] = B[i]`
  - Matmult `A[i][j] = B[i][k] * C[k][j]`
  - Transpose `A[i][j] = B[j][i]`

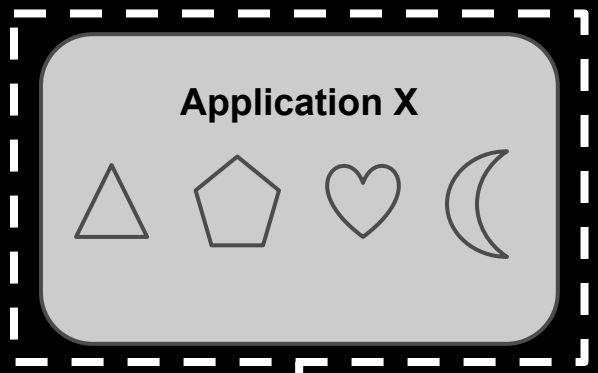
Benefits of Idiom-enabled Approach

- Idioms are fundamental
  - part of the application/problem, not the HW
- Idioms are concise
  - readily identifiable in source code
- Idioms are pervasive
  - you already know many of them! (matmult, stream)
  - well-understood as performance constructs
- Idioms are persistent
  - appear within many languages and platforms
  - common to many applications

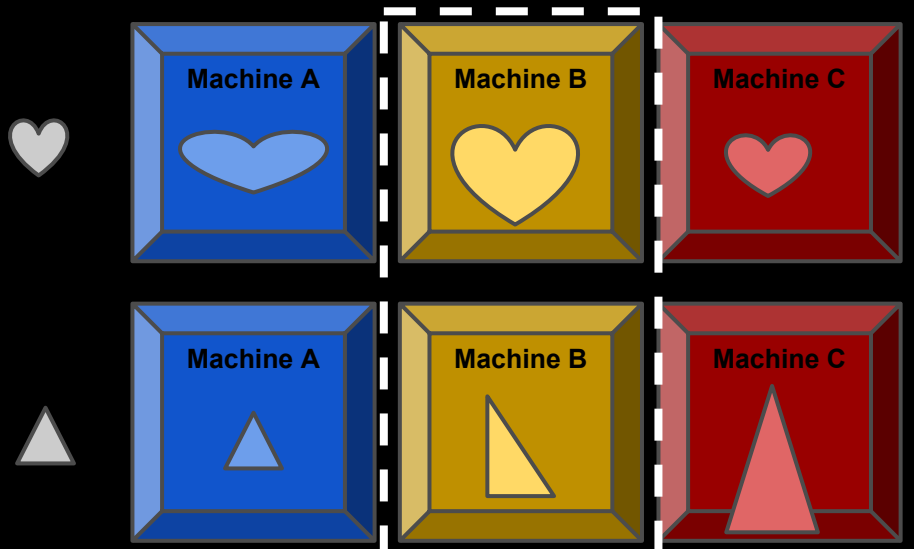
Idioms in Action

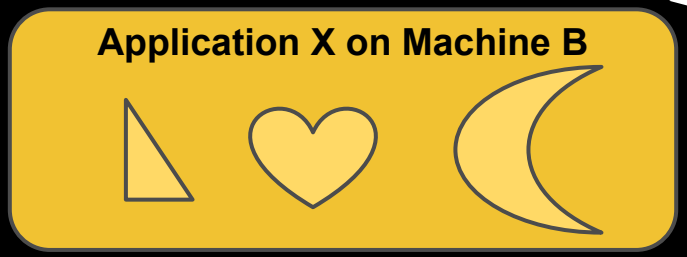
The Convolution Method

Step 1: Identify idioms in the application



Step 2: Characterize idioms on machines





Step 3: Convolve application and machine characterizations

The Idiom Hypotheses

- HPC applications can largely be described as (combinations of) idioms
  - Other information can be critical (working set size, access pattern, ILP, branchiness, many others)

- The set of idioms covering a significant fraction of HPC computation is small
  - On the order of 10-20

In Support of Using Idioms

- Goal - show that a small number of idioms cover a significant fraction of execution time

- Our approach
  1. Comb application looking for idioms within loops
  2. Detailed profiling of application to see where time is spent
  3. Attach detected idioms to most important loop profiles

Experimental Methodology

- DoD HPCMP TI13 Benchmarks
  - AVUS - Air Vehicles Unstructured Solver
  - HYCOM - HYbrid Coordinated Ocean Model
- Hardware platform
  - Diamond @ ERDC
  - Intel Nehalem processors
- Software tools used (free + open source)
  - PIR
    - Identifies idioms in source code
    - GCC plugin (many langs/platforms supported)
  - PEBIL
    - Static analysis and binary instrumentation

AVUS “Turret-td”, 1024 cores

- Selected loop profiles (idioms joined by hand)

File Line #	Function	Dyn Insn %	Depth	FPOp / Insn	DefUse Dist INT	L3 Cache HR %	Idiom(s)	Composed Idioms
karl6sc.F 45,145	karl6sc_	28.26	3	0.443	5.58	91.12	stream	Magnitude-6 dot products
walldst.F 47-57	walldst_	16.58	6	0.290	5.06	99.96	stream	Data dependent streams
pathos.F 33	pathos_	< 1.00	1	0.000	2.50	97.25	reduction	Simple reduction
int_vec.F 40-42	int_vec_	< 1.00	1	0.000	4.00	76.60	gather	Simple gather

- Encouraging data on idiom/loop coverage
  - Loops are the major component of computation - 86.23% of instructions are inside loops
  - Current idiom coverage is significant but needs work: ~1200 loops total, ~450 loops containing idioms are found

Hycom “Large”, 1001 cores

- Selected loop profiles

File Line #	Function	Dyn Insn %	Depth	FPOp / Insn	DefUse Dist INT	L3 Cache HR %	Idiom(s)
momtum.f 432-532	momtum_	15.30	2	0.474	7.04	88.05	stream, reduction
hybgen.f 1704-1776	hybgen_weno_coefs_	6.40	2	0.356	3.69	99.99	stream
cnuity.f 55-72	cnuity_	4.83	3	0.359	7.37	90.26	stream
tsadvc.f 809-840	advem_fct2_	3.63	3	0.295	10.8	94.56	stream
floats_ 707-714	mod_floats.F	< 1.00	2	0.000	5.00	100.00	transpose

- Encouraging data on idiom/loop coverage
  - 83.45% of instructions are inside loops
  - ~2000 loops total, ~600 loops containing idioms are found

The Challenges Ahead

- Coverage: identify complete idiom taxonomy
  - Currently ~10 identified
- Identify idioms on exotic platforms
  - Currently have a GCC plugin, but more is needed
- Matching source to binary line numbers
  - Compilers make this difficult (inlining, loop unrolling, code elimination, etc.)
  - Currently requires tedious human labor
- Composing idioms
  - Loops with interdependent idioms are the norm
  - Identifying them in application is straightforward
  - How do we reason about their behavior?

Bonus Slides

Further Reading

- *An Idiom-finding Tool for Increasing Productivity of Accelerators*, ICS 2011.
- *Modeling and Predicting Performance of High Performance Computing Applications*. JHPCA 2013.
- *PIR: PMaC's Idiom Recognizer*, ICPPW 2010.
- *A Methodology for Understanding Performance of Stencil Computations on Intel's Xeon Phi*, Cluster 2013.
- *Defining Software Requirements for Scientific Computing*, DARPA HPCS 2004.

- Level 1
  - Level 2
    - Level 3
-