Using Computational Patterns to Understand Heterogeneity

David Sheffield and Kurt Keutzer
and the PALLAS team
Michael Anderson, Bryan Catanzaro (→ Nvidia),
Katya Gonina, Chao-Yue Lai, Mark Murphy,
Bor-Yiing Su, Narayanan Sundaram

Motivation

- Future generations of microprocessesors are certain to have heterogeneous elements
- But determining the precise mix of heterogeneity is difficult:
  - Different microarchitectures to support the same ISA
    - Large and small cores
  - Significant coprocessor cores (eg, GPUs)
  - Special purpose execution units
    - ISA additions (eg Tensilica TIE)
    - Autonomous execution units (eg next-route lookup in network processors)
  - Reconfigurable logic
- Computational patterns give a new perspective on identifying heterogeneity to create new architectures

Approach

- We will review the computational patterns
- We will identify key program features to best support the computational patterns
- We will identify key micro-architectural elements that best support these program features
- Show implications of these micro-architectural elements on microprocessor-level heterogeneity

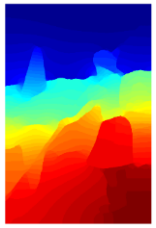
- Caveat: workload dependent: your mileage may vary

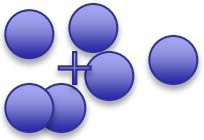
Computational patterns

Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Structured Grid												
Dense Matrix												
Sparse Matrix												
Spectral (FFT)												
Monte Carlo												
N-Body												

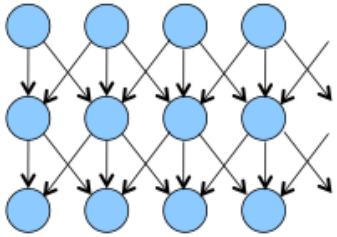
From applications to architectures

Applications





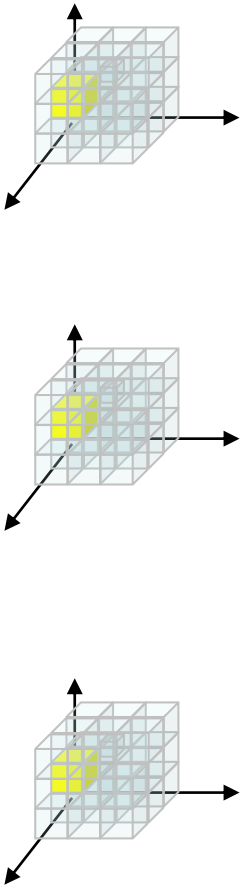
Pixel Dependencies



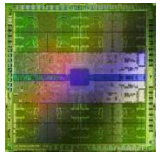
Computational Patterns

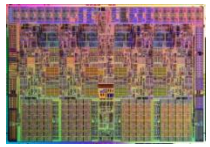
Computational patterns

Program features

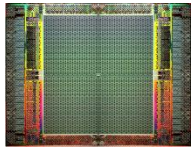


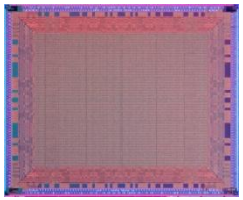
Architectures





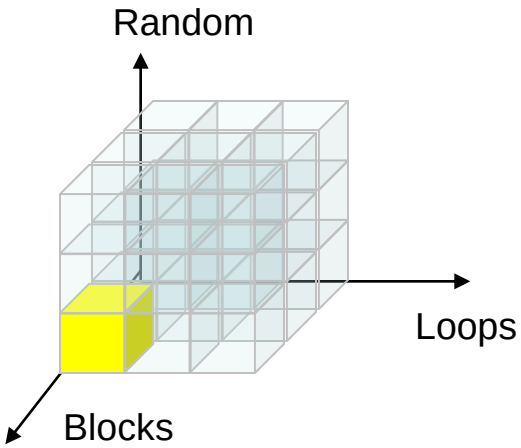
Accelerators



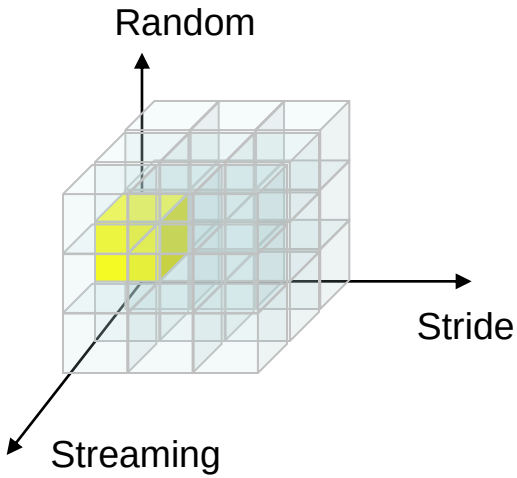


Program features

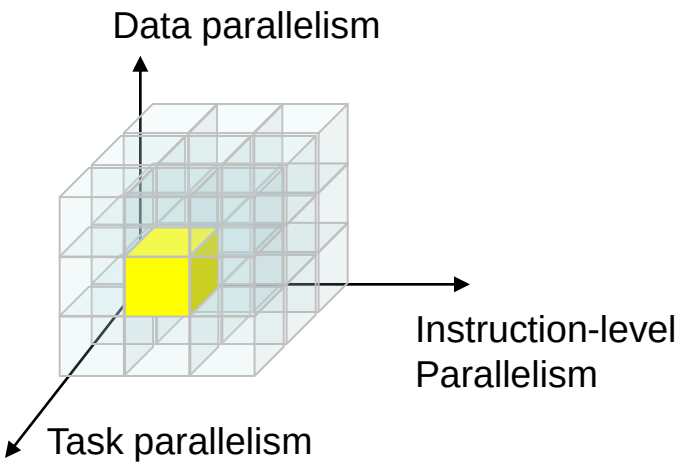
Instruction Access Pattern



Data Access Pattern

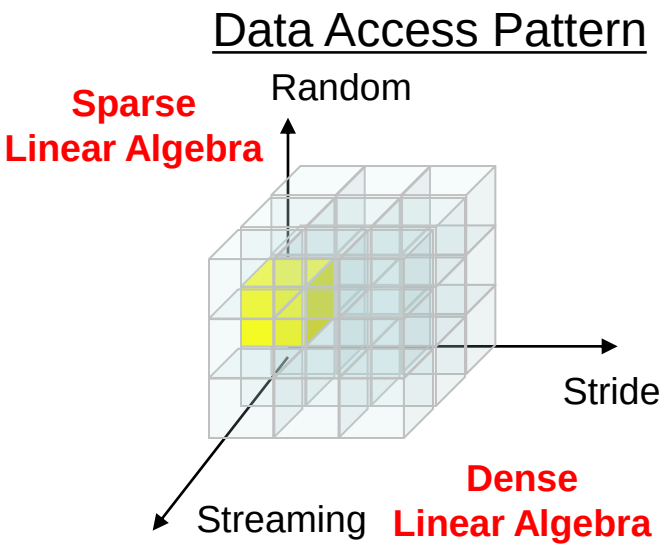


Concurrency Opportunities



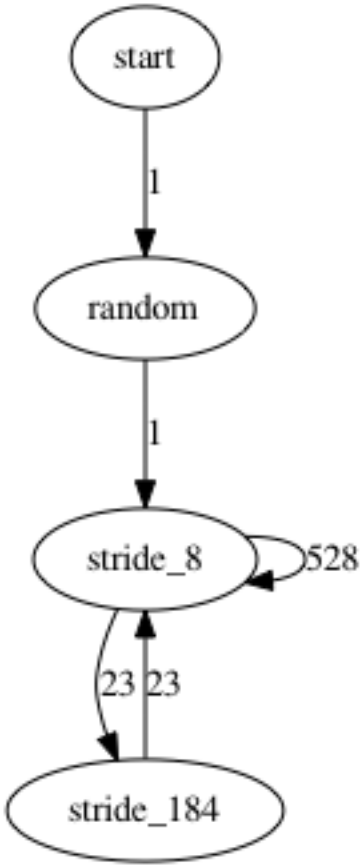
The developer needs to be aware of low-level application characteristics to efficient map applications onto heterogeneous platforms

Patterns from the machine's perspective

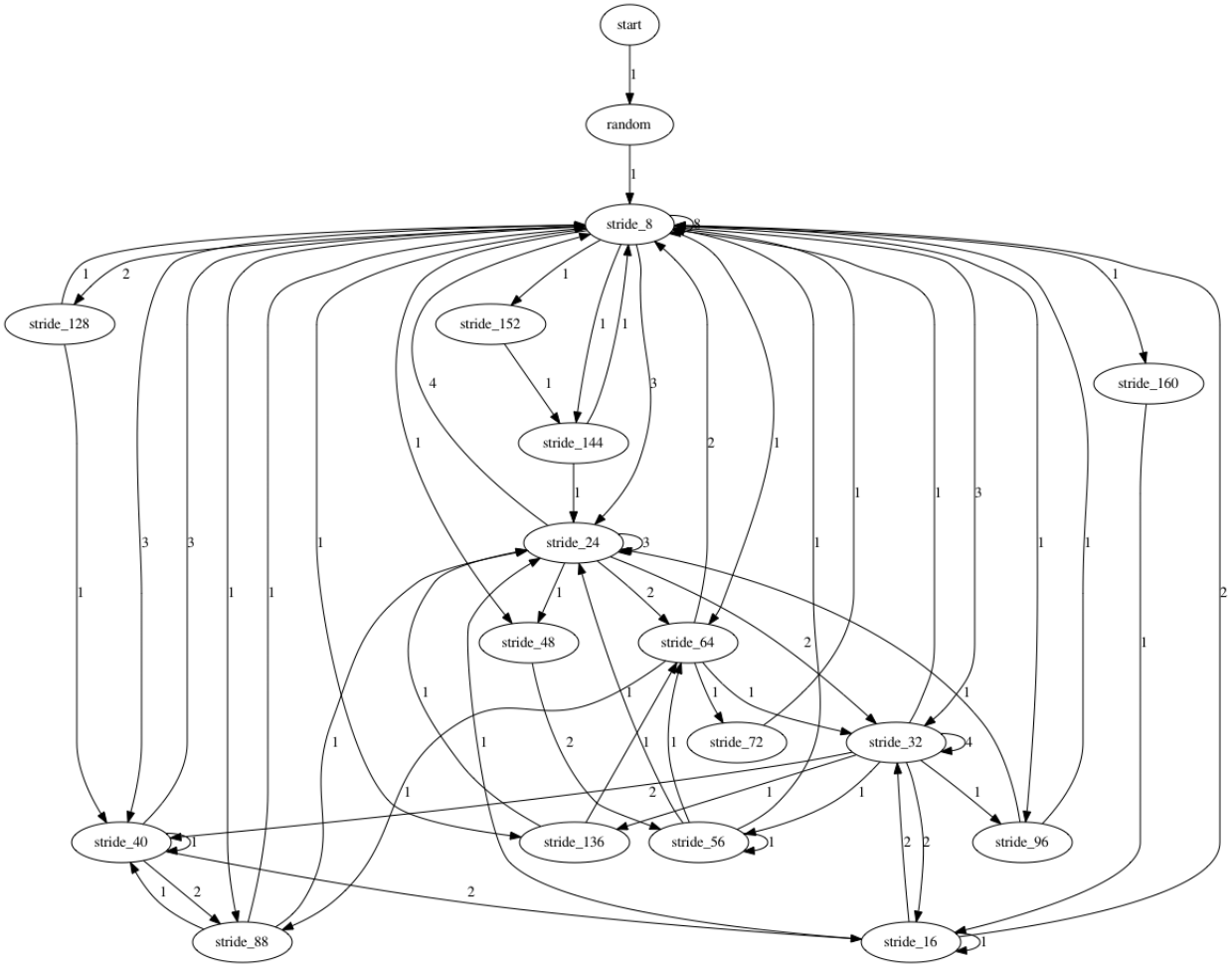


- The differences we observed between dense and sparse linear algebra can be mapped to the cube
  - Microarchitectural features can be optimized for the dimensions of the cube
    - DMA for streaming
    - Multithreaded processor for random accesses

Patterns from the machine's perspective



Dense linear algebra



Sparse linear algebra

Microarchitecture elements

- Memory subsystem
  - Caches
  - Software scratchpads
  - DMA engines
  - Coalescing hardware
- Processor core configuration
  - Issue width
    - Wide out-of-order vs scalar in-order
  - Data parallel support
    - SIMD width
    - multithreaded execution
- System configuration
  - Cache coherence
  - Message passing
  - NUMA
  - Distribution of core type

Preliminary Dwarf Chart

- We presented 9 program features earlier
- Instruction access patterns
  - Random
  - Blocks
  - Loops
- Data access patterns
  - Random
  - Stride
  - Streaming
- Concurrency opportunities
  - Data
  - Task
  - Instruction
- We are building a heatmap for program features
  - Reflects what we know today
  - Not complete

Preliminary Dwarf Chart

	Instruction access			Data access			Parallelism		
	Random	Blocks	Loops	Random	Strided	Streaming	Data	Task	Instruction
Dense linear algebra									
Sparse linear algebra									
Structured grids									
Unstructured grids									
Spectral methods									
Particle methods									
Monte Carlo methods									
Combinational logic									
Finite state machines									
Backtrack and B&B									
Graph algorithms									
Dynamic programming									

Conclusions

- Patterns help the developer determine “where does computation want to happen”
- From our study of existing applications, we believe patterns can guide exploration of applications on emerging heterogeneous platforms
- New heterogeneous platforms will make the algorithmic design space more complicated
  - A disciplined programming methodology is required to fully exploit these new platforms