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 microprocessors 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 (e.g., GPUs)
  - Special purpose execution units
    - ISA additions (e.g., Tensilica TIE)
    - Autonomous execution units (e.g., 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

<table>
<thead>
<tr>
<th>Apps</th>
<th>Embed</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>HPC</th>
<th>CAD</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack / B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prog.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monte Carlo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-Body</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From applications to architectures

Applications

Computational Patterns

Program features

Architectures

Pixel Dependencies

Computational patterns
Program features

Instruction Access Pattern

- Random
- Loops
- Blocks

Concurrency Opportunities

- Data parallelism
- Task parallelism
- Instruction-level Parallelism

Data Access Pattern

- Random
- Stride
- Streaming

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

<table>
<thead>
<tr>
<th>Instruction access</th>
<th>Data access</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>Blocks</td>
<td>Loops</td>
</tr>
<tr>
<td>Dense linear algebra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured grids</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured grids</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Particle methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monte Carlo methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combinational logic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finite state machines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack and B&amp;B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph algorithms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic programming</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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