How Programming Systems Meet the Needs of New Architecture Classes:

Past, Present and Future

Mary Hall
University of Utah
Collaborators and Acknowledgements

Stencils & Bricks
Tuowen Zhao, Sam Williams, Hans Johansen, Protonu Basu (Facebook), Brian Van Straalen, Phil Colella, Lenny Oliker

Sparse Tensors & Reprs.
Anand Venkat (Intel), Michelle Strout, Khalid Ahmad, Cathie Olschanowsky, Mahdi Mohammadi, Eddie Davis, Hari Sundar, John Jolly

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and by the National Science Foundation under CCF-1564074. This research used resources in Lawrence Berkeley National Laboratory and the National Energy Research Scientific Computing Center, which are supported by the U.S. Department of Energy Office of Science’s Advanced Scientific Computing Research program under contract number DE-AC02-05CH11231.
A Cycle That Repeats

Observe/Predict Technology Trend

New Architectures to Address Technology Trend

Period of Assessment

New Programming Systems Designed for New Architectures

Bell’s Law: A new computer class forms roughly each decade establishing a new industry (cheaper, smaller, more capable). Gordon Bell, CACM, Jan. 2007.
Recent Eras of Computing

Technology Trends Driving Each Decade of Innovation
1980s 1990s 2000s 2010s 2020s

FLOPS

Architecture Solutions
Vector/Multiprocessor Supercomputer e.g., Cray 2 (1985)

Programming System Solutions
Vectorizing compilers, Feedback to programmers on data dependences, Early parallelizing compilers
Recent Eras of Computing

Technology Trends Driving Each Decade of Innovation

1980s  | 1990s  | 2000s  | 2010s  | 2020s

FLOPS   | Memory Wall

Architecture Solutions

Killer Micros
Scalable Shared Memory
Multiprocessors
Complex Memory Hierarchies
In-Memory Computation

Programming System Solutions

Locality Optimization
Automatic Parallelization
Automatic Data Decomposition

Recent Eras of Computing

Technology Trends Driving Each Decade of Innovation

1980s 1990s 2000s 2010s 2020s

FLOPS Memory Wall End of Dennard Scaling

Architecture Solutions
Multi-Core Everywhere
Low Power and Power Management
SIMD Multimedia ISA Extensions

Programming System Solutions
PL Support for Expressing Thread ||ism
Vectorizing & Parallelizing Compilers, Revisited
Software Power Management
More Locality Optimization

Slide source: Kathy Yelick, HEPiX 2009
Data from Olukoton, Hammond, Sutter, Smith, Batten, Asanovic
Recent Eras of Computing

Technology Trends Driving Each Decade of Innovation
1980s 1990s 2000s 2010s 2020s

- FLOPS
- Memory Wall
- End of Dennard Scaling
- Scalability & Software Limitations

Architecture Solutions
Specialization to achieve efficiency
Throughput-oriented systems
Accelerators
Memory and storage technologies

Programming System Solutions
Domain-specific languages/libraries/tools
GPGPU
Autotuning
Compiler integration with libraries

Sarkar et al., Exascale Software Study, DARPA, 2009.
Recent Eras of Computing

Technology Trends Driving Each Decade of Innovation

1980s  |  1990s  |  2000s  |  2010s  |  2020s
---|---|---|---|---
FLOPS | Memory Wall | End of Dennard Scaling | Scalability & Software Limitations | Data Movement Dominant

Message: No longer can afford to waste time/energy on unnecessary data movement!

The 2020s Era:
Time to Get Serious about Reducing & Accelerating Data Movement
Solution: Co-Optimization

- Computation
- Data Layout & Data Representation
- Architecture-Specific Code Generation
1. Reducing Vertical Data Movement Through More Efficient Use of the Memory Hierarchy

- **Locality Optimization**: Access the same or nearby data in “fast” memory
- Typically through **reordering transformations** such as, e.g., **tiling (blocking)**
  - Reorder computation to change its memory access pattern

What if instead data is stored to match or improve its memory access pattern?
2. Accelerating Data Movement

Consider hardware features designed to maximize bandwidth or capitalize on spatial reuse
- DRAM page mode access
- Tensor Cores
- Wide SIMD
- Cache lines larger than a word
- TLBs translating addresses in the same page
- ...

What if data is organized to take advantage of these to increase efficiency of requisite data movement?
Example: Stencil Computation Pattern

- Stencils arise in PDE solvers
- Low order: memory bound
- High order: compute bound … but movement of intermediate data

\[
\text{Out}[i][j] = \text{coeff} \times (\text{In}[i][j-3] + \text{In}[i][j-2] + \text{In}[i][j-1] + \text{In}[i][j+1] + \text{In}[i][j+2] + \text{In}[i][j+3] + \text{In}[i-3][j] + \text{In}[i-2][j] + \text{In}[i-1][j] + \text{In}[i+1][j] + \text{In}[i+2][j] + \text{In}[i+3][j]);
\]
Vertical Data Movement Arising from Stencils

Impact of multiple address streams on data movement

- Capacity misses in caches and TLB
- Limits hardware prefetching effectiveness
- Reordering in registers

Many-core parallelism and tiling make this worse!

7 distinct address streams!
Co-Optimization for Stencils: Bricks

Brick Data Layout + Code Generator

• A brick is a mini (e.g., 8x8x8) subdomain without a ghost zone
• Application of a stencil reaches into other bricks (affinity important)
• Implemented with contiguous storage and adjacency lists

[Zhao et al., PP3HPC 2018] [Zhao et al., SC 2019]
[Zhao et al., PPoPP 2021]
3. Reducing On-Node Data Movement during Communication

Problem:

- Non-contiguous data to be communicated to neighbors
- Application may need to copy to/from buffers as part of communication
- Even with MPI library support (e.g., MPI Types), there is still data movement
- Additional data movement may occur between host and accelerator as part of communication

What if the *physical* organization of data is optimized for communication, but a different *logical* organization is exposed?
Packing ghost zones for communication can be as costly as sending the data on the interconnect, limiting strong scaling.

**Horizontal Data Movement Arising from Stencils**

- Communicate subset of node’s domain needed by other nodes
- May need to **pack** (i.e., reorganize) data as part of communication
- e.g., Ghost zones, domain surface

<table>
<thead>
<tr>
<th>Direction</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send North</td>
<td>{6,7,8}</td>
</tr>
<tr>
<td>Send South</td>
<td>{1,2,3}</td>
</tr>
<tr>
<td>Send East</td>
<td>{3,5,8}</td>
</tr>
<tr>
<td>Send West</td>
<td>{1,4,6}</td>
</tr>
<tr>
<td>Send Northeast</td>
<td>{6}</td>
</tr>
<tr>
<td>Send Northwest</td>
<td>{8}</td>
</tr>
<tr>
<td>Send Southwest</td>
<td>{1}</td>
</tr>
<tr>
<td>Send Southeast</td>
<td>{3}</td>
</tr>
</tbody>
</table>

Packing ghost zones for communication can be as costly as sending the data on the interconnect, limiting strong scaling.
Co-Optimization: Bricks Stored in Communication Order
4. Reducing Data Movement by Making Data Smaller

- **Optimizing Data Value Representation:** Modify data values to reduce size of data without sacrificing too much accuracy
  - Reduced or mixed precision floats
  - Sparsifying data
  - Compressing data

What if the domain-specific language or compiler provides abstractions for modifying data values, and verifying accuracy?
Co-Optimization: Mixed Precision SpMV

for (i=0; i<numrows; i++)
for (j=index[i]; j<index[i+1]; j++)
y[i] += A[j]*x[col[j]];

Approach
- Use single precision for values well-represented, range [-1,1]
- Use double precision for everything else

[Ahmad et al., TACO'19]
[Ahmad et al., HiPEAC'19]

Fig. 6. Performance speedup of mixed precision over double precision SpMV on V100 for all 2,802 sparse matrices.
Proposal to the Programming System Community

Build ecosystem of compiler dialects and code generators
What Will the Computers of the 2030s Look Like?

- Fundamentally different, e.g., quantum?
- Still exploring lots of different directions?
- Convergence a la the 1990s?