A Hybrid Systolic-Dataflow Architecture for Inductive Matrix Algorithms

Jian Weng, Sihao Liu, Zhengrong Wang, Vidushi Dadu, Tony Nowatzki
University of California, Los Angeles
Feb 27th, 2020
Inductive Kernels

• Inductive Kernels
  • QR, SVD, Cholesky, Solver

• Challenges
  • Poor vectorization
  • Too small to be multi-threaded
General Purpose Processor Performance

% of the ideal performance

dsp  cpu  gpu

% of the ideal performance

svd-12  svd-24  svd-32  qr-12  qr-24  qr-32  cho-12  cho-24  cho-32  sol-12  sol-24  sol-32  avg
Our Goal: An Efficient, Flexible, and Specialized Architecture

• Base Design: Spatial Architecture
• Specialize Inductive Idioms
  • Hybridizing PEs, Inductive Control, and Implicit Vector Padding
  • Scale up with multiple lanes
• REVEL: Reconfigurable Vector Lanes
  • 3.4x, and 17x speedup over existing spatial architectures, and general purpose processors
  • 2x power and half area comparing with ASIC collection
Outline

• **Background**
  - “Dataflow” or “Systolic”? A tradeoff between cost and flexibility
  - Challenge 1: Synchronous Coordination
  - Challenge 2: Overwhelmed Processing Elements
  - Challenge 3: Overwhelmed Coordination
    - Inductive Access
    - Padding the Vectorization

• **REVEL: Reconfigurable Vector Lanes**

• **Evaluation**
Spatial Architecture

- Architectures expose computing resource and on-chip network to programming interfaces
- Multiple instructions shared PE
- Dynamically scheduled execution

Each PE is dedicated to one instruction
- The timing of data arrival are determined when compilation

Tagged Dataflow

Systolic

Dependence Graph

5.8x Area
4.2x Power
Spatial Architecture Performance

% of the ideal performance

- systolic
- tagged dataflow

- svd-12
- svd-24
- svd-32
- qr-12
- qr-24
- qr-32
- cho-12
- cho-24
- cho-32
- sol-12
- sol-24
- sol-32
- avg
Base Design: Systolic Architecture with Decoupled Data Access

- Arithmetic operations are offloaded onto spatial architecture
- Data access are decoupled and coordinated by the controller
- Synchronization buffers serve as interfaces between dynamic/static timing
Challenge 1: Non-uniform Produce/Consume Rate

```c
for (j = 0; j < n; ++j) {
    x[j] = x[j] / A[j,j];
    for (i = j + 1; i < n; ++i)
        x[i] = x[j] * A[j,i];
}
```
for (k=0; k<n; ++k) {
    inv = 1.0 / a[k,k];
    invsqrt = 1.0 / sqrt(a[k,k]);
    for (j=k; j<n; ++j)
        l[j,k] = a[k,j] * invsqrt;
    for (j=k+1; j<n; ++j)
        for (i=j; i<n; ++i)
            a[j,i] = a[k,i] * a[k,j] * inv;
}
Challenge 3.1: Overwhelmed Coordination

```c
for (j=0; j<n; ++j) {
    x[j] = x[j]/a[j,j];
    for (i=j+1; i<n; i+=2) {
        x[i] -= x[j]*a[j,i];
    }
}
```

Rectangular Slicing:

Triangular Slicing:
Challenge 3.2: Imperfect Loop Tiling

```c
for (j=0; j<n; ++j) {
    x[j] = x[j]/a[j,j];
    for (i=j+1; i<n; i+=2) {
        x[i] -= x[j]*a[j,i];
        if (i+1<n) x[i+1] -= x[j]*a[j,i+1];
    }
}
```

Rectangular Slicing:
- `a[n:n, p:q]`

Triangular Slicing:
- `a[j+3:n]`
- `a[j+2:n]`
- `a[j+1:n]`
- `a[n-2:n]`
Outline

• Spatial Architecture

• REVEL: Reconfigurable Vector Lane
  • Specialization 1: Hybridizing processing elements
  • Specialization 2: Coordinating non-uniform dependences
  • Specialization 3: Inductive Access Intrinsics
  • Specialization 4: Implicit vectorization predication
  • Scalability: Larger Spatial or Multiple Lanes?

• Evaluation
Specialization 1: Hybridizing Systolic and Dataflow

for (k=0; k<n; ++k) {
    inv = 1.0 / a[k,k];
    invsqrt = 1.0 / sqrt(a[k,k]);
    for (j=k; j<n; ++j)
        l[j,k] = a[k,j] * invsqrt;
    for (j=k+1; j<n; ++j)
        for (i=j; i<n; ++i)
            a[j,i] = a[k,i] * a[k,j] * inv;
}
Specialization 2: Coordinating Non-uniform Dependences

for (j=0; j<n; ++j) {
    x[j] = x[j]/a[j,j];
    for (i=j+1; i<n; ++i)
        x[i] = x[i] - x[j]*a[j,i];
}
Specialization 3: Inductive Memory Access

\[
\text{for (j=0; j<n; ++j) \{ }
\]
\[
x[j] = x[j]/a[j,j];
\]
\[
\text{for (i=j+1; i<n; i+=2) \{ }
\]
\[
x[i] -= x[j]*a[j,i];
\]
\[
\text{if (i+1<n) x[i+1] -= x[j]*a[j,i+1];}
\]
\[
\text{\}}
\]
Specialization 4: Implicit Vectorization Predication

for (j=0; j<n; ++j) {
    x[j] = x[j]/a[j,j];
    for (i=j+1; i<n; i+=2) {
        x[i] -= x[j]*a[j,i];
        if (i+1<n) x[i+1] -= x[j]*a[j,i+1];
    }
}

Generate masks according to the striding pattern

Triangular Slicing:
- a[n-2:n]
- a[j+3:n]
- a[j+2:n]
- a[j+1:n]
Scalability: A larger mesh or multiple lanes?
REVEL: Reconfigurable Vector Lanes

- A centralized control core commands multiple lanes
  - Predication based broadcast
  - SIMT-like control commands
- Each lane is independent to execute
Outline

• Spatial Architecture
• REVEL: Reconfigurable Vector Lanes
• Evaluation
  • Methodology
  • Speedup
Evaluation Methodology

- **Performance**
  - Gem5 RISC-V in-order core integrated with a cycle-accurate spatial architecture simulator
    - Extending the stream-dataflow ISA
  - Baselines:
    - Intel(R) Xeon(R) Silver 4116 @2.10GHz (Intel MKL)
    - TI6678 DSP @1.25GHz (TI DSPLIB)
    - NVIDIA Titan (cuBlas)

- **Power/Area**
  - Spatial Architecture implemented in Chisel
  - Synthesized in Synopsys DC 28nm @1.25GHz
  - SRAM power/area are estimated by CACTI.
Speedup over the TI DSP (log scale)

- **CPU**
- **GPU**
- **Systolic**
- **Tagged**
- **Revel**

The graph shows the speedup comparison among different processors for various benchmarks. The x-axis represents different benchmarks (svd-32, qr-32, cho-32, sol-32, fir-199x1024, mm-48x16-64, fft-1024, gm) and the y-axis represents the speedup over the TI DSP on a log scale.
Speedup (Batch-1)

Speedup Over the TI DSP (log scale)

cpu | gpu | systolic | dataflow | revel

svd-32 | qr-32 | cho-32 | sol-32 | fir-199x1024 | mm-48x16-64 | fft-1024 | gm

[Bar chart showing speedup over the TI DSP for different benchmarks and hardware platforms.]
Conclusion

• According to our results, REVEL and its hybrid architecture is a promising next-generation digital signal processing architecture.

• More broadly, our work demonstrates the importance of considering multiple execution models.