ElasticFlow: A Complexity-Effective Approach for Pipelining Irregular Loop Nests

Mingxing Tan¹ ², Gai Liu¹, Ritchie Zhao¹, Steve Dai¹, Zhiru Zhang¹

¹ Computer Systems Laboratory, Electrical and Computer Engineering, Cornell University

² Google Inc.
Outline

- Loop pipelining in HLS
- Irregular loop nest
- ElasticFlow architecture
- ElasticFlow synthesis
- Experimental results
An important optimization in HLS
- Create a static schedule for the loop body to allow successive loop iterations to be overlapped

**Objective**

Initiation Interval (II) ↓
Throughput ↑

```c
for(i=0; i < 4; i++)
    for(j=0; j < 4; j++)
    {
        #pragma pipeline
        acc += A[j] * i;
    }
```
Pipelining Outer Loop

1. Pipelining only inner loop

```
for(i=0; i < 4; i++){
    for(j=0; j < 4; j++){
        acc += A[j] * i;
    }
}
```

1 inner loop iteration per cycle

2. Pipelining outer loop by unrolling inner loop

```
#pragma pipeline
for(i=0; i < 4; i++){
    acc += A[0] * i;
    acc += A[1] * i;
}
```

1 outer loop iteration per cycle

Fixed inner loop bound
Contains one or more dynamic-bound inner loops
- Number of inner loop iterations vary during runtime
- Accesses less-regular data structures (e.g. sparse matrices, graphs, and hash tables) common in emerging applications
- How to pipeline this loop nest to achieve one lookup per cycle?

```c
for (i : keys_to_find)
#pragma pipeline
    hv = Jenkins_hash(k);
    p = hashtbl[hv].keys;
    while (p && p->key!=k)
        p = p->next;
    format_output(p)
}
```
Aggressively Unrolling Inner Loop

for (i : keys_to_find)
#pragma pipeline

hv = Jenkins_hash(k);
p = hashtbl[hv].keys;

#pragma unroll

for (j=0; j<6; j++)
if (p && p->key!=k)
p = p->next;

format_output(p)

}
Issues with Aggressive Unrolling

1. May not be statically determinable

2. Worst-case bound >> common case (e.g. 99 vs. 2)

3. Unnecessarily deep pipeline, very inefficient in area
Need for a New Approach

- Irregular loop nests are prevalent
  - Graph processing,
  - Scientific computation,
  - Image processing, etc.

- Naive approaches result in low throughput or large area
  - Need resource-efficient pipelining of the outer loop for an irregular loop nest to target one outer loop iteration per cycle
ElasticFlow Concept

- **ElasticFlow**
  - Architecture and associated synthesis techniques
  - Effectively accelerate irregular loop nests

- **Transform the irregular loop nest into a multi-stage dataflow pipeline**
  - Dynamically distribute different outer loop instances of the dynamic-bound inner loop to one or more processing units
  - Inner loops execute in a pipelined fashion across different outer loop iterations
ElasticFlow Architecture

- Each dynamic-bound inner loop is mapped to an application-specific *loop processing array* (LPA)
  - LPA contains one or more *loop processing units* (LPUs)
  - Each LPU executes an inner loop until completion, which automatically handles inner loop carried dependences
Distributor and Collector

- **Distributor**
  - Dynamically distributes inner loop instances to LPUs

- **Collector**
  - Collects results from the LPUs
  - Acts as an reorder buffer (ROB) to ensure that results are committed to the next stage in-order
ElasticFlow on Hash Lookup

for (i : keys_to_find) {
    #pragma pipeline
    hv = Jenkins_hash(k);
    p = hashtbl[hv].keys;
    while (p && p->key!=k)
        p = p->next;
    format_output(p)
}

Dynamically overlap inner loops across outer loop iterations to achieve a throughput of one outer loop iteration per cycle
Execution with Single LPU

- Single LPU for Stage B
  - Execution in Stage A and C can overlap in time
  - Inner loop iterations execute serially on Stage B

```c
for (i : keys_to_find) {
    #pragma pipeline
    hv = Jenkins_hash(k);
    p = hashtbl[hv].keys;
    while (p && p->key!=k) {
        p = p->next;
    }
    format_output(p)
}
```

Throughput bottlenecked by the inner loop latency in stage B.
Execution with Multiple LPUs

- Multiple LPUs for Stage B
  - Dynamically schedule inner loops

Multiple LPUs for B

Single LPU for B
Dynamic Scheduling

- Dynamic scheduling policy
  - Mitigates the effect of unbalanced latency variation of different inner loops

Inefficient resource utilization due to many stalls and idles!
Multiple Dynamic-Bound Inner Loops

for (i=0; i<num_keys; i++)
#pragma pipeline

// A: lookup hashtbl1
...

// B: dynamic-bound loop
while (p && p->key!=k)
    p = p->next;

// C: loop up hashtbl2
...

// D: dynamic-bound loop
while (q && q->key!=k)
    q = q->next;

// E: merge results
...

Database join

Architecture with dedicated LPAs

Each LPA is dedicated to a particular inner loop
Issues with Dedicated LPAs

- If loop B incurs much longer average latency than loop D, the LPA for loop D results in poor resource utilization.
LPA Sharing

- An LPA can be shared among one or more inner loops
  - sLPU: single-loop processing unit, dedicated to one loop
  - mLPU: multi-loop processing unit, shared among multiple loops
  - sLPA: single-loop processing array, consists of multiple sLPUs for a particular loop
  - mLPA: multi-loop processing array, consists of multiple mLPUs each shared among loops

Architecture with shared LPUs vs. sLPA
mLPA improves resource utilizations and performance by reducing pipeline stalls for unbalanced workload.

Execution of `dbjoin` on dedicated LPAs:

- **sLPA^B**:
  - sLPU_1: i^B=0, i^D=0
  - sLPU_2: i^B=1, i^D=1
  - sLPU_3: i^B=2, i^D=2
  - sLPU_4: i^B=3, i^D=4
- **sLPA^D**:
  - sLPU_1: i^B=2, i^D=3
  - sLPU_2: i^B=5, i^D=4
  - sLPU_3: Idle

Execution on shared mLPA:

- **mLPA^{B,D}**:
  - mLPU_1: i^B=4, i^D=2
  - mLPU_2: i^B=1, i^D=1
  - mLPU_3: i^B=3, i^D=5
  - mLPU_4: Idle

Even requires fewer LPUs.
ElasticFlow Synthesis

- Maps irregular loop nest to the ElasticFlow architecture
  - Partition the loop nest into multiple stages
  - Identify inner loop candidates to form the LPAs
  - Synthesize these inner loops into sLPUs and mLPUs

```c
for (i=0; i<num_keys; i++)
  #pragma pipeline
  
  // A: lookup hashtbl1
  ...
  // B: dynamic-bound loop
  while (p && p->key!=k)
    p = p->next;
  // C: loop up hashtbl2
  ...
  // D: dynamic-bound loop
  while (q && q->key!=k)
    q = q->next;
  // E: merge results
  ...
```
sLPU Allocation

Definitions

- **TP**: Expected number of outer loop iterations per cycle
- **II<sub>i</sub>**: Achievable initiation interval (II) of inner loop <i>i</i>
- **L<sub>i</sub>**: Latency in cycles of a single iteration of loop <i>i</i>
- **B<sub>i</sub>**: Common-case bound of inner loop <i>i</i> (from profiling)

Number of sLPUs

Common-case latency of each inner loop instance

Need this many sLPU to hide the latency of inner loop

To achieve the expected throughput

How many simultaneous in-flight outer loop iterations is required?

\[
U_i = \left[ II_i (B_i - 1) + L_i \right] \cdot TP
\]
mLPU Allocation

- Replace dedicated sLPUs with shared mLPUs to improve performance and resource utilization
  - How many sLPUs should be replaced with mLPUs?

- Inherent trade-off between performance and area
  - mLPUs improve performance by allowing adaptive assignment of resources to different types of loops depending on workload
  - mLPUs typically consume more area than sLPUs
LPU Allocation

- Optimize the tradeoff as an integer linear program given
  - Resource usage of each type of LPU
  - Area of the sLPA architecture

\[
\text{maximize } \alpha \sum_{k=1}^{K} \sum_{i=1}^{N} D_k r_{ik} + \beta \sum_{k=1}^{K} n_k \quad \text{subject to}
\]

\[
\sum_{k=1}^{K} S_k^j n_k \leq A_{total}^j \quad \forall j
\]

- Total area of the LPAs

\[
\sum_{i=1}^{N} U_i r_{ik} \geq n_k \quad \forall k
\]

- Prevent over-allocation of LPUs

\[
\sum_{k=1}^{K} r_{ik} = 1 \quad \forall i
\]

- Each loop maps to a single type of LPA

\[
r_{ik} = 0 \quad \forall k \notin T(i)
\]

- Loops mapped to compatible LPA
**ROB Buffer Sizing**

- Reorder buffer (ROB) must hold all results from the LPUs that are not yet ready to be committed
  - Distributor stalled when ROB is full. LPUs cannot process new outer loop iterations, and become underutilized.

Need to store results from i=1 to i=7 because they finish before i=0

---

Problem: how to statically but suitably size the ROB during synthesis?
We estimate the ROB size based on profiling

- Maximum latency $L_{\text{max}}$
- Minimum latency $L_{\text{min}}$
- Average latency $L_{\text{avg}}$
- Latency standard deviation $\sigma$

Our estimates (for K LPUs) achieve good performance based on the following empirical formulation

$$S = \frac{L_{\text{max}}}{\max(L_{\text{avg}} - 3\sigma, L_{\text{min}})}(K - 1) + 1$$
Deadlock Avoidance

- Both sLPA and mLPA are deadlock-free
  - Limit the number of in-flight outer loop iterations to be no greater than the number of available ROB entries

- Entire dataflow architecture cannot deadlock
  - If the architecture forms a directed acyclic graph
  - If there is data dependence between shared inner loops
Experimental Setup

- ElasticFlow’s setup leverages a commercial HLS tool which uses LLVM compiler as its front-end

- Compared ElasticFlow to pipelining techniques employed in state-of-the-arts commercial HLS tool

- Target Xilinx Virtex-7 FPGA with 5-ns target clock period

- Benchmark applications
  - Graph processing, database, scientific computing, image processing
Performance for Different Number of LPUs

Close to proportional improvement in performance for increasing number of LPUs
### ElasticFlow vs. Aggressive Unrolling

- Achieves comparable performance with significantly less resource usage
  - Unrolling is inapplicable when the worst-case loop bound cannot be statically determined

<table>
<thead>
<tr>
<th>Design</th>
<th>Technique</th>
<th>Latency</th>
<th>LUTs</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbjoin</td>
<td>Unroll</td>
<td>386</td>
<td>10632</td>
<td>21187</td>
</tr>
<tr>
<td></td>
<td>ElasticFlow</td>
<td>389</td>
<td>6493</td>
<td>4239</td>
</tr>
<tr>
<td>spmv</td>
<td>Unroll</td>
<td>365</td>
<td>2884</td>
<td>6319</td>
</tr>
<tr>
<td></td>
<td>ElasticFlow</td>
<td>372</td>
<td>1894</td>
<td>1412</td>
</tr>
</tbody>
</table>

- 1.5x reduction
- 4.5x reduction
Effectiveness of LPU Sharing

- Using mLPA can further improve the performance by 21%-34% with similar area

Comparison of mLPUs over sLPUs

<table>
<thead>
<tr>
<th>Design</th>
<th># sLPUs</th>
<th># mLPUs</th>
<th>Latency Reduction</th>
<th>Slice Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfd-A</td>
<td>8</td>
<td>8</td>
<td>34.7%</td>
<td>3.8%</td>
</tr>
<tr>
<td>cfd-B</td>
<td>16</td>
<td>16</td>
<td>31.5%</td>
<td>5.2%</td>
</tr>
<tr>
<td>dbjoin-A</td>
<td>8</td>
<td>7</td>
<td>21.3%</td>
<td>7.0%</td>
</tr>
<tr>
<td>dbjoin-B</td>
<td>16</td>
<td>14</td>
<td>21.6%</td>
<td>5.7%</td>
</tr>
</tbody>
</table>

Significant latency reduction
Small area overhead
Take-Away Points

- Existing HLS tools rely on static pipelining techniques
  - Extract parallelism only at compile time
  - Not competitive for irregular programs with dynamic parallelism

- Need for adaptive pipelining techniques
  - Dynamically extract parallelism at runtime
  - Efficiently handle statically unanalyzable program patterns

- We address pipelining of irregular loop nests containing dynamic-bound inner loops
  - Novel dataflow pipeline architecture and synthesis techniques
  - Substantial performance improvement
ElasticFlow: A Complexity-Effective Approach for Pipelining Irregular Loop Nests

Mingxing Tan\textsuperscript{1,2}, Gai Liu\textsuperscript{1}, Ritchie Zhao\textsuperscript{1}, Steve Dai\textsuperscript{1}, Zhiru Zhang\textsuperscript{1}

\textsuperscript{1} Computer Systems Laboratory, Electrical and Computer Engineering, Cornell University

\textsuperscript{2} Google Inc.
Backup Slides
Coarse-Grained Pipelined Accelerators (CGPA)

- Liu, Johnson, and August, DAC’14
- Generates coarse-grained pipelines for a loop nest by partitioning it into parallel and non-parallel sections
  - Employs replicated data-level parallelism to create multiple identical copies of the parallel section
  - Applies decoupled pipeline parallelism to separate the parallel and sequential sections with a set of FIFOs
- ElasticFlow achieves additional performance and resource efficiency
  - Enables out-of-order execution and dynamic scheduling
  - Optimizes allocation and sharing of LPUs with mLPA architecture
  - Studies sizing for both ROB and delay line and runtime policy to prevent deadlock
Comparison with CGPA

![Comparison with CGPA](image-url)
Kocberber, Grot, Picorel, Falsafi, Lim, and Ranganathan, MICRO’13

A reconfigurable accelerator for hash indexing in database systems
  - Uses decoupled pipeline architecture similar to ElasticFlow
  - Hashing unit distributes work to a parallel array of walker units, whose results are combined in a \( n \) output unit

ElasticFlow is a technique for addressing a more general problem of pipelining irregular loop nests