Dynamo: A Runtime Codesign Environment

Heather Quinn\textsuperscript{1}, Miriam Leezer
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
{hquinn, mel}@ece.neu.edu

L. A. Smith King
Dept. of Mathematics and Computer Science
College of the Holy Cross
Worcester, MA 01610
LA@cs.holycross.edu

Systems using Field Programmable Gate Array (FPGA) boards have been proven effective for gaining one to three orders of magnitudes speedup over systems based solely on PCs. Signal and image processing applications are especially attractive for implementation on FPGAs as their computationally intensive and massively parallel algorithms can effectively take advantage of the FPGA architecture. In moving part of an algorithm to hardware, one must consider overhead costs as well as the improvement in the computation to determine whether the move will result in overall system speedup. Dynamo is a runtime system for generating hardware/software pipeline implementations. Dynamo balances the benefits of hardware and software implementations and takes overhead costs into account in order to accurately predict runtimes of hardware/software systems.

Currently, the Dynamo system generates hardware/software solutions for image processing applications from a library of predefined component implementations. Many image processing applications consist of a series of algorithms applied to the image, forming a computation pipeline. When using Dynamo, an image analyst only needs to specify the pipeline of image processing components to apply and an image or images to process. Image processing components are chosen from a library of predefined components: the image processing Basic Library of Components (ipBLOC). Each component in the library has at least one software and one hardware implementation associated with it. From the pipeline specification, Dynamo selects the most efficient combination of hardware or software component implementations to minimize pipeline runtime (pipeline assignment), generates the source code for a HW/SW implementation of the pipeline (pipeline compilation), processes the input image(s) using the generated pipeline (pipeline execution), and returns the result to the analyst.

![Figure 1: System Overview](image)

The goals of the Dynamo system are to allow an image analyst to focus solely on pipeline selection, create image processing pipelines at runtime, make efficient use of software and FPGA hardware, and build pipelines that maximize performance. Currently, Dynamo is restricted to image processing applications, but is easily extensible to other application domains. Dynamo’s design includes three major subsystems which implement pipeline assignment, pipeline compilation and pipeline execution, respectively. These subsystems

\textsuperscript{1}Heather Quinn is supported under a Graduate Student Researchers Program Fellowship through NASA’s Goddard Space Flight Center.
**Report Documentation Page**

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

<table>
<thead>
<tr>
<th>1. REPORT DATE</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 FEB 2005</td>
<td>N/A</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamo: A Runtime Codesign Environment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5a. CONTRACT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5b. GRANT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5c. PROGRAM ELEMENT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5d. PROJECT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5e. TASK NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5f. WORK UNIT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHOR(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department of Electrical and Computer Engineering Northeastern University Boston, MA 02115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10. SPONSOR/MONITOR’S ACRONYM(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11. SPONSOR/MONITOR’S REPORT NUMBER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DISTRIBUTION/AVAILABILITY STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release, distribution unlimited</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13. SUPPLEMENTARY NOTES</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>14. ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15. SUBJECT TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16. SECURITY CLASSIFICATION OF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. REPORT unclassified</td>
</tr>
<tr>
<td>b. ABSTRACT unclassified</td>
</tr>
<tr>
<td>c. THIS PAGE unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17. LIMITATION OF ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>UU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18. NUMBER OF PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19a. NAME OF RESPONSIBLE PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
interact with Dynamo’s predefined libraries, including the ipBLOC. Figure 1 shows an overview of the Dynamo system.

At the core of the Dynamo runtime system is pipeline assignment (PA). PA assigns hardware or software implementations to each component in the pipeline to minimize total pipeline execution time, while meeting the area requirements for the FPGA device. There are costs associated with passing intermediate images and data between components that differ depending on component types and interfaces. These costs include data movement, communication, reprogramming and hardware initialization. PA must balance the latency and area costs to find the best solution for a given FPGA device. Profiles of the component implementations and of overhead runtimes allow PA to accurately predict pipeline solution runtimes. The pipeline assignment problem is NP-complete, and is similar to the hardware/software co-design partitioning problem.

The Dynamo subsystem responsible for solving PA is named SHARPP (Software/Hardware Runtime Procedural Partitioning). SHARPP solves the pipeline assignment problem quickly at runtime and uses different algorithms for different pipeline sizes to keep execution time short. SHARPP uses dynamic programming for small pipelines (1-7 components) and two variations of tabu search to solve large pipelines (7-20 components). Tabu search is a metaheuristic based on local search which provides near-optimal PA solutions. The two variations find solutions that are on average 18% slower than the optimal pipeline latency.

Pipeline compilation and execution occur within the runtime environment. Dynamo’s RIPS (Runtime Interfacing for Pipeline Synthesis) subsystem performs pipeline compilation. RIPS uses the SHARPP output to compose an executable that calls the appropriate implementations and overhead methods that comprise a pipeline solution. The pipelines are constructed at runtime because pre-contracting PA solutions for all possible combinations of component implementations is not practical. Finally, the pipeline is executed. Dynamo’s execution system runs the compiled pipeline on the selected image and outputs the results.

As an illustration of how Dynamo works, assume the image analyst has chosen to run the pipeline “median filter → histogram” on a 40165-pixel image. Table 2 shows the four possible solutions to this pipeline and the SHARPP predicted latencies. The component assignments are shown within parentheses with “sw” indicating software and “hw” indicating hardware. These results show that the optimal solution for this image size uses the hardware implementations for both components. Figure 2 shows what the pipeline looks like when the overhead costs are included in the pipeline, where the squares represent the components and the circles represent the overhead methods. RIPS was used to build executable solutions for each of the four pipelines. These pipelines were executed and measured so the actual execution times could be compared with SHARPP’s estimates. The relative error for the predicted latency is quite low.

We have tested pipelines that range in size from one to 20 components. Many short pipelines are completely assigned to software since the hardware initialization cost is relatively high. As pipelines included more components, enough speedup will be gained by processing in hardware to mitigate the device initialization cost. Therefore, longer pipelines tend to be mainly assigned to hardware, except when either the image is small or the component is not well matched to the FPGA architecture.

![Figure 2: Pipeline Solution with Overhead Methods](image)

<table>
<thead>
<tr>
<th>Annotated Solution</th>
<th>Predicted Runtime (ms)</th>
<th>Actual Runtime (ms)</th>
<th>Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>(median:sw) → (histogram:sw)</td>
<td>3766</td>
<td>3801</td>
<td>0.009</td>
</tr>
<tr>
<td>(median:sw) → (histogram:hw)</td>
<td>5076</td>
<td>5844</td>
<td>-0.038</td>
</tr>
<tr>
<td>(median:hw) → (histogram:sw)</td>
<td>2772</td>
<td>2836</td>
<td>0.023</td>
</tr>
<tr>
<td>(median:hw) → (histogram:hw)</td>
<td>2718</td>
<td>2260</td>
<td>-0.169</td>
</tr>
</tbody>
</table>

Table 2: The Four Solutions to median filter → histogram
Dynamo: A Runtime Codesign Environment

Heather Quinn1, Dr. Miriam Leeser
Northeastern University

Dr. Laurie Smith King
College of the Holy Cross

Motivation: Accelerate image processing tasks through efficient use of FPGAs. Combine already designed components at runtime to implement series of transformations (pipelines)

Fast, Flexible Image Processing

Run this pipeline:

On this Environment:

Which component implementations to use?
How to minimize overall latency?
When to use FPGA?
How to change the pipeline or interfaces dynamically?

Goal: If pipeline selection is left to the image analyst, can the other three steps be performed automatically at runtime?

Pipeline Selection:
- Choosing and ordering components

Pipeline Assignment:
- Assigning pipelines to minimize overall latency with the efficient use of software and FPGA

Pipeline Compilation:
- Creating image processing pipelines dynamically

Pipeline Execution:
- Executing image processing pipelines

SW/HW Runtime Procedural Partitioning Tool

Solves PA within either fixed or adaptive time limit based on user's choice
Chooses an algorithm to solve PA based on pipeline size

Optimization Method

Fixed
Adaptive

Dynamic Programming

1-5
1-15

Opt Tabu Search with Greedy

8.9

Opt Tabu Search with All Hardware

10-20
16-20

Runtime Interfacing for Pipeline Synthesis

Builds executable pipeline from PA solution
Connects the appropriate implementations so that the coupling costs are satisfied

Packet Exchange Platform

Runtime Environment’s Communication Agent

Separates pipeline from runtime environment
Makes communication abstract and generic

Future Work

- Extend the pipeline assignment problem for FPGA devices
  - in a network of workstations
  - with embedded processors
- Extend the pipeline assignment problem's objectives to include power minimization
- Extend the latency model to include an estimation of the error for better accuracy

Reconfigurable Systems

Using reconfigurable hardware incurs execution costs not present in software or ASIC-based systems
- Hardware initialization
- Communication
- Reprogramming

Efficient Use of FPGAs

Software algorithms' runtime for small images less than the hardware costs
- Profiling the hardware and software runtimes for different image sizes determines the crossover point
- Deciding at runtime to execute in software or hardware is simple for one algorithm processing one image

Image Processing Pipelines

Series of image processing algorithms applied to an image
Each algorithm has a software and hardware implementation
Finding the optimal pipeline assignment is complicated
- Exponential number of implementations
- Coupling costs differ for each pipeline assignment
- Need a strategy to find a fast pipeline implementation at runtime

SW/HW Runtime Procedural Partitioning Tool

Solves PA within either fixed or adaptive time limit based on user's choice
Chooses an algorithm to solve PA based on pipeline size

Optimization Method

Fixed
Adaptive

Dynamic Programming

1-5
1-15

Opt Tabu Search with Greedy

8.9

Opt Tabu Search with All Hardware

10-20
16-20

A Two Component Pipeline

Median Filter

Image size of 40185 pixels

Latency w/o overhead

Actual Latency

ARE* w/ overhead

Median Filter

4905
2516
4509
2509
2516

Histogram

2894
370
2884
35

A Two Component Pipeline

Median Filter

Image size of 40185 pixels

Latency w/o overhead

Actual Latency

ARE* w/ overhead

Median Filter

4905
2516
4509
2509
2516

Histogram

2894
370
2884
35

Hardware initialization
Communication
Reprogramming

Expensive Use of FPGAs

Software algorithm's runtime for small images less than the hardware costs
- Profiling the hardware and software runtimes for different image sizes determines the crossover point
- Deciding at runtime to execute in software or hardware is simple for one algorithm processing one image

Efficient Use of FPGAs

Software algorithm's runtime for small images less than the hardware costs
- Profiling the hardware and software runtimes for different image sizes determines the crossover point
- Deciding at runtime to execute in software or hardware is simple for one algorithm processing one image

Pipeline Assignment:
- Assigning pipelines to minimize overall latency with the efficient use of software and FPGA

Pipeline Compilation:
- Creating image processing pipelines dynamically

Pipeline Execution:
- Executing image processing pipelines

Four Shortcomings in Codesign

- Applications are configured statically
- Design is not sensitive to user changes
- FPGA-based tools do not account for overhead costs
- Latency is underestimated
- Partition bound too early
- Interface changes too costly
- System code needs extensive rewrites

Four Challenges to Codesign

- Unified implementation languages
- Partitioning design
- The pipeline assignment problem
- Interfacing hardware and software
- Abstract communication layer and runtime interface synthesis
- Choosing a target architecture
- One FPGA and one GPP

Future Work

- Extend the pipeline assignment problem for FPGA devices
  - in a network of workstations
  - with embedded processors
- Extend the pipeline assignment problem’s objectives to include power minimization
- Extend the latency model to include an estimation of the error for better accuracy

Publications


Dynamo: A Runtime Codesign Environment

Dr. Heather Quinn, Dr. Miriam Leeser
Northeastern University
hquinn, mel@ece.neu.edu

Dr. Laurie Smith King
College of the Holy Cross
la@mathcs.holycross.edu

On this Environment:

Which implementations to use?
How to minimize overall latency?
When to use FPGA?
How to change the pipeline or interfaces dynamically?

Run this pipeline:

Median Filter & Edge Detection
Goals

Predict pipeline latency accurately:
Model includes all overhead costs of using the FPGA

Implement low latency solutions:
Problem space quickly searched

Build and execute pipelines at runtime:
User is not impacted
Dynamo System Overview

**Input:** The pipeline specification, an image, and a device name

**Output:** Results from executing pipeline on the image

**Libraries:** ipBLOC

**Model:** Pipeline Assignment