Multicore DSP Software Synthesis using Partial Expansion of Dataflow Graphs

George F. Zaki, William Plishker, Shuvra S. Bhattacharyya
University of Maryland, College Park, MD, USA

&

Frank Fruth
Texas Instruments Inc., Germantown, MD, USA

Presented at the Workshop on Software Synthesis, Montreal, Canada, October 4, 2013.

Presentation version: October 8, 2013
Outline

• Motivation & background
• Partial expansion of SDF graphs
• Online mapping using PEGs (partial expansion graphs)
  – RTOS-based code generation and implementation
• Experimental evaluation
• Summary
Motivations

• Complex Dynamic applications:
  – LTE, Video processing.

• Exposing appropriate form(s) and amount of parallelism:
  – Task, Data and Pipeline.

• Exponential complexity scheduling problem.

• Enabling offline analysis, runtime optimization and automatic code generation.
Two Approaches

Classical Approach
- **Re-write** your kernels (using new parallel algorithms) and programming models (e.g., CUDA, OpenMP).
- Customized efficient implementation.
- Programming models are targeted towards very specific platforms.

Model Based Approach
- Represent your application in terms of an appropriate high level model and write a separate platform model.
- Use tools to analyze and schedule your implementation.
- **Re-use** your previous investments in optimized kernels and developed systems.
Model Goals

- Adequately represent three sources of parallelism in order to efficiently utilize the underlying architecture.

Sources of parallelism:
- Data parallelism
  → utilizing production and consumption rates in synchronous dataflow (SDF)
- Task parallelism (implicit in expanded DFG)
- Pipeline parallelism
  (software-pipelined looped schedules)
SDF Scheduling

• An SDF graph $G = (V, E)$ has a “valid schedule” if it is deadlock-free and is sample rate consistent (i.e., it has a periodic schedule that fires each actor at least once and produces no net change in the number of tokens on each edge) [Lee 1987]

• For each actor $v$ in a connected, consistent SDF graph, there is a unique repetition count $q(v)$, which gives the number of times that $v$ must be executed in a minimal valid schedule.

$q(A) = 3 \quad q(B) = 2$
Outline

• Motivation & background
• Partial expansion of SDF graphs
• Online mapping using PEGs (partial expansion graphs)
  – RTOS-based code generation and implementation
• Experimental evaluation
• Summary
Classical SDF Multiprocessor Scheduling

- Fully expand the SDF graph to its equivalent HSDF graph (DAG)
- … then pass the generated single-rate graph to a scheduling algorithm.

\[ q(A) = 1, \quad q(B) = M, \quad q(C) = M^2 \]

a) Original multirate SDF graph.

b) Full expansion to a directed acyclic graph.

Exponential expansion.
Partial expansion of SDF graphs

- A partial expansion graph (PEG) $G_p = (V_p, E_p)$ is an undirected graph that is used for scalable mapping of an SDF graph $G$ onto a programmable platform.
- The vertex set is defined as $V_p = I_p \cup \{B\}$, where $I_p$ corresponds to a set of instances.
- Each such instance executes a group of successive actor firings.
- $B$ is a special inter-vertex coordination actor called the buffer manager of the PEG.
- $E_p = \{\{B, i\} \mid i \in I_p\}$ — i.e., an (undirected) edge is connected between the buffer manager and every other vertex in the PEG.

a) Original multirate SDF graph.

b) Possible partial expansion.
PEG (continued)

- Every actor $v$ in $G$ is instantiated $N_v$ times in $G_p$ where $N_v$ depends on the relative load of $v$ in the graph in terms of execution time.
- $N_v \leq N_b$ where $N_b$ is the number of processor cores.
- Each actor with internal state is instantiated exactly once (i.e., $N_v = 1$).
- Distinct instances of the same actor are mapped to different cores.
- … and can execute concurrently.
- Using the PEG, we exploit multiple levels of graph parallelism.
  - Data parallelism through partial expansion.
  - Task parallelism using the SDF graph.
  - Pipeline parallelism by overlapping the execution of multiple graph iterations.
Related Work

• Classical scheduling of HSDF Graphs --- e.g., [Kwok, 1999] and many others
  – Require full dataflow graph expansion
• Adaptive work stealing [Tzannes 2010]
  – Focuses on task and data parallelism
  – Can be integrated with our buffer manager approach
• Adjustment of actor granularity (clustering) based on statically known execution time estimates (e.g., [Gordon 2006][Kudlur 2008])
  – Cannot adapt scheduling structures over time if execution characteristics change
• Pragma/annotation-based distribution of dynamic activations at run-time (e.g., [Bellens 2006][Blagojevic 2007])
  – In contrast, our PEG-based approach applies model-based specifications in terms of dataflow graphs
Outline

• Introduction & background

• Contributions
  – Partial expansion of SDF graphs
  – Buffer manager & dynamic scheduling
  – PEG online mapping.
  – Code generation using RTOS
  – Evaluation.

• Conclusions & future work
Buffer Manager (BM)

- The buffer manager is the process that coordinates the sharing of state across instances that share the same SDF graph actors,
- ... and coordinates data transfer between communicating instances that are mapped to different processors.
- The buffer manager schedules the activation of instances by sending them special data packets called PEG messages.
- A PEG message encapsulates dataflow information and pointers to one or more memory blocks that implement the SDF graph edges.
- In the PEG, buffers are decomposed into slots, where each slot has a fixed capacity (maximum number of tokens).
Dynamic Scheduling

- Useful for actors that have data dependent execution times.
- The buffer manager assigns activations to instances of a partially expanded actor upon completion of previous firings on which the activations depend.
- Assignment and acknowledgement messages ping-pong between the buffer manager and the instances.
Outline

• Motivation & background
• Partial expansion of SDF graphs
  • Online mapping using PEGs (partial expansion graphs)
    – RTOS-based code generation and implementation
• Experimental evaluation
• Summary
Automatic Scheduling of PEG

- A PEG solution consists of two parts:
  1. By how much should each actor should be expanded?
  2. To which core is each instance mapped?

- We apply a two-stage scheduling approach:
  - Calculate the amount of expansion of each actor using particle swarm optimization (PSO).
  - Map each instance to a core using a mapping heuristic.
  - Generate code for a generic implementation that can be customized at run-time through dynamic adaptation of mappings.
PEG Scheduling Workflow

1. Initialize Swarm Particles
2. Expansion Solution
3. Map Instances to Cores
4. PEG solution
5. Update Swarm
6. Fitness Function
7. Customize Generic Implementation
Particle Swarm Optimization

- A particle in a PEG optimization problem is a vector that holds the partial expansion value of every actor in the dataflow graph.
- The swarm consists of $N$ particles, where in every iteration, particles don’t “die”, but rather they change their positions.

```
P1  1  3  2
Shown expansion

P2  1  1  1
No expansion
```
Particle Swarm Optimization (cont.)

Next position

Swarm’s best

Particle’s best

local_inertia

global_inertia

current_influence

\[
local_{inertia}_p[v] = \text{rand()} \times (best_p[v] - s_p[v])
\]

\[
global_{inertia}_p[v] = \text{rand()} \times (best_S[v] - s_p[v])
\]

\[
s_p[v] = s_p[v] + C1 \times local_{inertia}_p[v] + C2 \times global_{inertia}_p[v]
\]
PEG Mapping Heuristic

- The PEG mapping heuristic takes as input the original SDF graph, the amount of expansion given from the PSO, and the platform to map the instances to cores.
- Its objective is to take advantage of the exposed amount of parallelism.
- Since PEGs are mainly targeting data parallel applications, priority in scheduling instances is ordered by data, task and then pipeline.
Mapping Heuristic Example

- Identify Delay Parallel Regions (DPR) using the $b$-value (bottom level) of actors.
- Start scheduling the DPR Leader (DPRL) and then subsequently other instances and actors within the same DPR.

![Diagram showing the mapping of actors to cores and the DPR structure]
Outline

• Motivation & background
• Partial expansion of SDF graphs
• Online mapping using PEGs (partial expansion graphs)
  – RTOS-based code generation and implementation
• Experimental evaluation
• Summary
Software Synthesis

- The PEG code generation module augments a real time OS (e.g., DSP/BIOS) with dataflow semantics to schedule and run instances in the PEG.
- Every instance is generated using a software interrupt (SWI) thread that is triggered upon reception of its PEG message.
- Once an execution of an instance activation completes, the thread notifies the buffer manager by with an acknowledgment message.
- The buffer manager then updates the status of the buffers connected to this instance,
- … and applies a first acknowledge, first assign (FAFA) algorithm to schedule new instances.
Integration with DSP/BIOS

• Initialization phase:
  – Queues for every instance are created.
  – IPC messages between the buffer manager and the instances are initialized
  – Every instance is associated with an SWI that will be triggered upon reception of the PEG message.
  – Vectorization is used to help amortize the interrupt overhead

• At Runtime:
  – The buffer manager assigns activations to instances.
  – Instances execute their work functions upon allocation of space in input and output buffers.
Evaluation on Texas Instruments EVM6472 Platform

- A customizable generic solution is implemented and used at run-time, based on dynamically changing activation patterns.
- In this solution, instances for every actor in the graph are implemented.
- Only instances that belong to a given solution are activated.
Evaluation on Texas Instruments EVM6472 Platform

• A customizable generic solution is implemented and used at run-time, based on dynamically changing activation patterns.
• In this solution, instances for every actor in the graph are implemented.
• Only instances that belong to a given solution are activated.

Example 1: A is expanded in 2 instances, while C is configured as a single instance.
Evaluation on Texas Instruments EVM6472 Platform

• A customizable generic solution is implemented and used at run-time, based on dynamically changing activation patterns.
• In this solution, instances for every actor in the graph are implemented.
• Only instances that belong to a given solution are activated.

Example 2: $A$ is configured as a single instance, while $C$ is expanded into 3 instances.
Outline

• Motivation & background
• Partial expansion of SDF graphs
• Online mapping using PEGs (partial expansion graphs)
  – RTOS-based code generation and implementation
• Experimental evaluation
• Summary
Evaluation: 1-Mapping Heuristic

- All experiments were conducted on a Texas Instruments TMS320C6472 six-core programmable digital signal processor.
- **One core is reserved to run the buffer manager thread.**
- The pseudo communication receiver benchmark [Zaki 2012] is used as a synthetic benchmark to measure speedups of different sources of parallelism and PEG scheduler overhead.

Pseudo-communication receiver application.
Speedups for Different Loads

- Task Graph: 44% of the execution load is in the parallel branches.
- Data Graph: 81% of the load is from a partially expanded actor.
- Pipeline Graph: All actors have the same load.
Evaluation: 1-Mapping Heuristic (continued)

Two real application dataflow graphs were used to test the mapping heuristic and explore the design spaces:

1. Digital communication receiver
2. Image registration

Digital communication receiver

Image registration
## Image registration

<table>
<thead>
<tr>
<th>Unrolled Iterations</th>
<th>Speedup</th>
<th>sift_r expansion</th>
<th>sift_t expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.20x</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4.17x</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4.64x</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4.76x</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4.70x</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

## Digital Communication benchmark

<table>
<thead>
<tr>
<th>Unrolled Iterations</th>
<th>Runtime(K cycles) (50 iterations)</th>
<th>Speedup</th>
<th>decoder expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>345022</td>
<td>1.92x</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>226967</td>
<td>2.92x</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>202638</td>
<td>3.27x</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>202646</td>
<td>3.27x</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>202644</td>
<td>3.27x</td>
<td>4</td>
</tr>
</tbody>
</table>
Evaluation: 2- Dynamic Scheduling

- The Boyer Moore actor is used to measure the effectiveness of FAFA scheduling compared to conventional methods for dispatching activations.

Intrusion detection application using the Boyer-Moore actor.
Evaluation: 2- Dynamic scheduling cont.

- Depends on the target string position when a match is found.
- The maximum standard deviation at runtime is achieved when the matching string can be located at any position within the packet, and
- … zero standard deviation means that all matching strings are located in the middle of the packet.
- 33% improvement over round robin distribution.
Evaluation: 3-Particle Swarm Optimization

2 * 4 mp-sched benchmark: this represents a solution space of 390625 possibilities for partial expansion only (without mapping).

mp-sched benchmark [GNU Radio]
Evaluation: 3-Particle Swarm Optimization

- Two particles of the swarm are initialized to explore the no-expansion and full-expansion solutions, while the others are initialized randomly.
- These results demonstrate the ability of the PSO to find effective levels of expansion for different levels of application load.
- The results take into account different levels of relative scheduling overhead.

<table>
<thead>
<tr>
<th>Total load (K cycles)</th>
<th>Runtime(K cycles) (50 iterations)</th>
<th>Speedup</th>
<th>filters expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000000</td>
<td>188374</td>
<td>2.65x</td>
<td>All filters → 5</td>
</tr>
<tr>
<td>50000</td>
<td>29867</td>
<td>1.67x</td>
<td>P1a &amp; P2a → 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>All others → 3</td>
</tr>
<tr>
<td>15000</td>
<td>16663</td>
<td>0.9x</td>
<td>All filters → 1</td>
</tr>
</tbody>
</table>
Evaluation: 4-Buffer Manager overhead

• For the pseudo-communication benchmark, the application actors have an execution time load \( \lambda \), which is assumed to be a multiple of the scheduling load \( S \).
• Actor-level vectorization (block processing) can be used to adjust the computation to scheduling ratio for a given application.
• The horizontal axis represents the ratio of \( \lambda \) to \( S \) for different scenarios.
Outline

• Motivation & background
• Partial expansion of SDF graphs
• Online mapping using PEGs (partial expansion graphs)
  – RTOS-based code generation and implementation
• Experimental evaluation
• Summary
Conclusions 1

• We present the Partial Expansion Graph, a new intermediate model and associated implementation strategy
  – Addresses the potential for exponential growth of SDF graph expansions.
• Task, data, and pipeline parallelism are represented and applied together to optimize performance.
• Code generation for RTOS-based implementation is developed.
• We have also developed automatic scheduling techniques that target PEG-based implementation using PSO and a PEG mapping heuristic.
Conclusions 2

• Reasonable dynamic scheduling speedup is achieved when the scheduling to computation ratio is 7:1.
• Variable-runtime applications exhibit a speedup of up to 33% compared to conventional methods for distributing activations.
• PSO provides significant speedups by
  – efficiently searching the design space,
  – allowing for control over local search spaces,
  – and balancing the computation-to-scheduling ratio.

Future work:
• Use of hardware support for buffer manager implementation.
• Analyze effect of buffer size on the application throughput for PEG-based implementation of dynamic applications.