The SEEC Framework and Runtime System

Henry Hoffmann
MIT CSAIL
http://heartbeats.csail.mit.edu/

High Performance Embedded Computing Workshop
September 21-22, 2011

This work was funded by the U.S. Government under the DARPA UHPC program. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.
In the beginning...*

Application programmers had one goal:

Performance

*The beginning, in this case, refers to the beginning of my career (1999)
But Modern Systems Have Increased the Burden on Application Programmers

Even worse, constraints can change dynamically
E.g. power cap, workload fluctuation, core failure
Most Programming Models Designed for Performance

Coherent Shared Memory

- Global Shared Cache
- Local Cache
- Data
- Store
- Network Interface
- Load

Message Passing

- Local Memory
- Data
- Store
- Network Interface
- Load

Concurrency

- Multi-threaded
- Multi-process

Communication

- Through Memory
- Through Network

Coordination

- Locks
- Messages

Control

- Procedural
- Procedural

Procedural control insufficient to meet the needs of modern systems
SEEC Replaces Procedural Control with Self-Aware Control

Procedural Control

- Run in open loop
- Assumptions made at design time
- Based on guesses about future

- Application optimized for system
- No flexibility to adapt to changes

Self-Aware Control

- Run in closed loop
- Understand user goals
- Monitor the environment

+ System optimizes for application
+ Flexibly adapt behavior

The self-aware model allows the system to solve constrained optimization problems dynamically
Outline

• Introduction/Motivation

The SEEC Model and Implementation

• Experimental Validation

• Conclusions
The SELF-awarE Computing (SEEC) Model

- **Goal:**
  Reduce programmer burden by continuously optimizing online

- **Key Features:**
  1. **Decoupled Approach:**
     - Applications explicitly state goals and progress
     - System software and hardware state available actions
     - The SEEC runtime system dynamically selects actions to maintain goals
  2. **General and Extensible:**
     - New applications can be supported without training
     - New actions can be added without redesign and reimplementation
Example Self-Aware System
Built from SEEC

Video Encoder

30 b/s

Goals:
30 beat/s,
Minimize Power

Control and Learning System

Observe

Desired Heart Rate

Observed Heart Rate

Error

Speedup

Application

SEEC
Controller

Act

Actuators

Algorithm

Cores

Frequency

Bandwidth

Decide

Observe

Act
### Roles in SEEC’s Decoupled Model

<table>
<thead>
<tr>
<th>Observe</th>
<th>Decide</th>
<th>Act</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Developer</td>
<td>Systems Developer</td>
<td>SEEC Runtime System</td>
</tr>
<tr>
<td>Express application goals and progress (e.g. frames/second)</td>
<td>Read goals and performance</td>
<td>Determine how to adapt (e.g. How much to speed up the application)</td>
</tr>
<tr>
<td>Provide a set of actions and a callback function (e.g. allocation of cores to process)</td>
<td>Initiate actions based on results of decision phase</td>
<td></td>
</tr>
</tbody>
</table>
Registering Application Goals

- **Performance**
  - Goals: target heart rate and/or latency between tagged heartbeats
  - Progress: issue heartbeats at important intervals
- **Quality**
  - Goals: distortion (distance from application defined nominal value)
  - Progress: distortion over last heartbeat
- **Power**
  - Goals: target heart rate / Watt and/or target energy between tagged heartbeats
  - Progress: Power/energy over last heartbeat interval

Research to date focuses on meeting performance while minimizing power/maximizing quality.
Registering System Actions

Each action has the following attributes:

- **Estimated Speedup**
  - Predicted benefit of taking an action

- **Cost**
  - Predicted downside of taking an action
  - Axis for cost (accuracy, power, etc.)

- **RPC handle**
  - A function that takes an id and implements the associated action
  - This is currently subject to change/redesign
The SEEC Decision Engine
(A general, extensible approach)

- Pros: Simple, Analyzable, Works well for profiled applications
- Cons: Lack of generality for unseen applications

**Classic Control System**: Generalized second order system

![Diagram of Classic Control System with Controller, Actuator, and Application blocks connected by arrows indicating performance goal and current performance.]
The SEEC Decision Engine
(A general, extensible approach)

- Pros: Adapts to unseen applications
- Cons: Assumes (relative) system models are correct
  Cannot support race-to-idle

Adaptive Control:
Based on 1-Dimensional Kalman Filter for Workload Estimation

Classic Control System

Performance Goal

Controller (Decide)

Actuator (Act)

Application (Observe)

Application Model (Decide)

Current Performance
The SEEC Decision Engine
(A general, extensible approach)

- **Pros**: Supports race-to-idle and proportional allocation
- **Cons**: May overprovision due to system model errors

Adaptive Action Selection:
Approximates solution to linear programming problem for meeting goals and minimizing cost

**Adaptive Control**

**Classic Control System**

Performance Goal

- **Controller (Decide)**
- **Actuator (Act)**
- **Application (Observe)**

Resource Model (Decide)

Application Model (Decide)
The SEEC Decision Engine
(A general, extensible approach)

Machine Learner:
Uses reinforcement learning to estimate system models online, becomes control system when models converge

Adaptive Action Selection

Adaptive Control

Classic Control System

Performance Goal

Controller (Decide)

Actuator (Act)

Application (Observe)

System Model (Decide)

Resource Model (Decide)

Application Model (Decide)
Outline

• Introduction/Motivation

• The SEEC Model and Implementation

Experimental Validation

• Conclusions
### Systems Built with SEEC

<table>
<thead>
<tr>
<th>System</th>
<th>Actions</th>
<th>Tradeoff</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Loop Perforation</td>
<td>Skip some loop iterations</td>
<td>Performance vs. Quality</td>
<td>7/13 PARSECs</td>
</tr>
<tr>
<td>Dynamic Knobs</td>
<td>Make static parameters dynamic</td>
<td>Performance vs. Quality</td>
<td>bodytrack, swaptions, x264, SWISH++</td>
</tr>
<tr>
<td>Core Scheduler</td>
<td>Assign N cores to application</td>
<td>Compute vs. Power</td>
<td>PARSEC</td>
</tr>
<tr>
<td>Clock Scaler</td>
<td>Change processor speed</td>
<td>Compute vs. Power</td>
<td>PARSEC</td>
</tr>
<tr>
<td>Bandwidth Allocator</td>
<td>Assign memory controllers to application</td>
<td>Memory vs. Power</td>
<td>STREAM, PARSEC</td>
</tr>
<tr>
<td>Power Manager</td>
<td>Combination of the three above</td>
<td>Performance vs. Power</td>
<td>PARSEC, STREAM, simple test apps (mergesort, binary search)</td>
</tr>
<tr>
<td>Learned Models</td>
<td>Power Manager with speedup and cost learned online</td>
<td>Performance vs. Power</td>
<td>PARSEC</td>
</tr>
<tr>
<td>Multi-App Control</td>
<td>Power Manager with multiple applications</td>
<td>Performance vs. Power and Quality for multiple applications</td>
<td>Combinations of PARSECs</td>
</tr>
</tbody>
</table>
## Systems Built with SEEC

<table>
<thead>
<tr>
<th>System</th>
<th>Actions</th>
<th>Tradeoff</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Loop Perforation</td>
<td>Skip some loop iterations</td>
<td>Performance vs. Quality</td>
<td>7/13 PARSECs</td>
</tr>
<tr>
<td>Dynamic Knobs</td>
<td>Make static parameters dynamic</td>
<td>Performance vs. Quality</td>
<td>bodytrack, swaptions, x264, SWISH++</td>
</tr>
<tr>
<td>Core Scheduler</td>
<td>Assign N cores to application</td>
<td>Compute vs. Power</td>
<td>PARSEC</td>
</tr>
<tr>
<td>Clock Scaler</td>
<td>Change processor speed</td>
<td>Compute vs. Power</td>
<td>PARSEC</td>
</tr>
<tr>
<td>Bandwidth Allocator</td>
<td>Assign memory controllers to application</td>
<td>Memory vs. Power</td>
<td>STREAM</td>
</tr>
<tr>
<td>Power Manager</td>
<td>Combination of the three above</td>
<td>Performance vs. Power</td>
<td>PARSEC, STREAM, extra test apps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(mergesort, binary search)</td>
</tr>
<tr>
<td>Learned Models</td>
<td>Power Manager with speedup and cost learned online</td>
<td>Performance vs. Power</td>
<td>PARSEC</td>
</tr>
<tr>
<td>Multi-App Control</td>
<td>Power Manager with multiple applications</td>
<td>Performance vs. Power and Quality for multiple applications</td>
<td>Combinations of PARSECs</td>
</tr>
</tbody>
</table>
Constrained Optimization: Managing Performance/Watt for PARSEC

Optimize performance/Watt on multiple machines

<table>
<thead>
<tr>
<th>Application Goals</th>
<th>System Actions</th>
<th>Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintain performance, minimize power</td>
<td>Allocate cores</td>
<td>Execute on two machines (w/ different power profiles)</td>
</tr>
<tr>
<td>Allocate clock speed</td>
<td>Allocate memory bandwidth</td>
<td>Compare SEEC to several other approaches including a static oracle</td>
</tr>
</tbody>
</table>
SEEC beats the static oracle by adjusting to phases within an application and recognizing when to race-to-idle.
SEEC is able to beat the static oracle on a different machine without code changes.
Learning Models Online

<table>
<thead>
<tr>
<th>Application Goals</th>
<th>System Actions</th>
<th>Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize power consumption while meeting target performance</td>
<td>Change cores, clock speed, and mem. bandwidth</td>
<td>Benchmark: STREAM</td>
</tr>
<tr>
<td>Initial models are incredibly optimistic (Assume linear speedup with any resource increase)</td>
<td></td>
<td>Observe convergence time and performance/Watt for converged system</td>
</tr>
</tbody>
</table>
SEEC Can Learn Models Online

SEEC learns not to allocate too many compute resources to a memory bound application
When System Models Are Wrong
(Breakdown)

Adaptive Control achieves performance fastest, but wastes power.
ML reaches target performance slowest, but saves power
Managing Application and System Resources Concurrently

Manage multiple applications when clock frequency changes

**Application Goals**

- **bodytrack**: maintain performance, minimize power
- **x264**: maintain performance, minimize quality loss

**System Actions**

- Change core allocation to both applications
- Change x264’s algorithms

**Experiment**

Maintain performance of both applications when clock frequency changes
SEEC Management of Multiple Applications
In Response to a Power Cap

bodytrack

- SEEC allocates cores to bodytrack

x264

- SEEC removes cores from x264

Clock drops 2.4-1.6GHz

w/o SEEC app misses goals

w/o SEEC app exceeds goals

SEEC adjusts algorithm to meet goals
Outline

• Introduction/Motivation

• The SEEC Framework

• Experimental Validation

Conclusions
Conclusions

- SEEC is designed to help ease programmer burden
  - Solves resource allocation problems
  - Adapts to fluctuations in environment and application behavior

- SEEC has two distinguishing features
  - Decoupled Design
    - Incorporates goals and feedback directly from the application
    - Allows independent specification of adaptation
  - General and Extensible Decision Engine
    - Uses an adaptive second order control system to manage adaptation

- Demonstrated the benefits of SEEC in several experiments
  - Optimize performance per Watt for multiple benchmarks on multiple machines
  - Adapts algorithms and resource allocation as environment changes
Thanks

- DARPA’s UHPC Program
- MIT, Politecnico di Milano, Freescale
- Martina Maggio, Marco D. Santambrogio, Jason E. Miller, Jonathan Eastep
- Stelios Sidiroglou, Sasa Misailovic, Michael Carbin
- Anant Agarwal, Martin Rinard, Alberto Leva, Jim Holt