SEEC: A Framework for SEIf-awarE Computing

Henry Hoffmann, Martina Maggio
Outline

Introduction/Motivating Example

• The SEEC Framework

• Experimental Validation

• Conclusions
Multicore Computing Systems Increase Burden on Application Developers

Today, application programmers have to address many, sometimes competing, concerns.

- Correctness
- Speed: Algorithms
- Speed: Architecture
- Power/Energy
- Quality

Today, application programmers have to address many, sometimes competing, concerns.
Example: Developing a Multicore Video Encoder

Allocate resources for best case

Allocate resources for worst case

Encoder must drop frames to keep up

Encoder Exceeds Goals

The power is low

The power is high, resources wasted

Application programmers need to balance competing constraints in fluctuating environments
Self-aware (or self-*, adaptive, autonomic, etc.) computing has become a discipline unto itself:

- Laddaga [DARPA BAA 1997, IEEE Intelligent Systems 1999]
- Kephart and Chess [IEEE Computer 2003]
- Babaoglu et al. [LNCS 2005]
The Self-Aware Computing Idea

Traditional Systems

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

- Programmer optimizes for system
- No flexibility to adapt to changes

Self-Aware Systems

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

- System optimizes for application
- Flexibly adapt behavior
Prior Work in Self-Aware Systems

- Self-aware/Adaptive/Autonomic systems have been used to solve problems in:
  - Software [Salehie & Tahvildari ACM TAAS 2009]
  - Real-time Systems [Block et al. ECRTS 2008]
  - Mobile Computing [Masters MobileHCI 2008]
  - Dynamic Compilation [Sorber et al, SenSys 2007, Baek & Chilimbi PLDI 2010]
  - Numerical Libraries [ATLAS, SPIRAL, FFTW]
  - Many others…

We build on previous work to create a programming model for self-aware systems
Outline

- Introduction/Motivating Example
- The SEEC Framework
  - Experimental Validation
  - Conclusions
SElf-awarE Computing (SEEC) Framework

- **Goal:**
  Reduce programmer burden with self-aware programming model
- **Key Features:**
  1. Applications *explicitly* state goals, system meets goals optimally
  2. One unified decision engine adapts algorithms, software and hardware
Example Self-Aware System
Built from SEEC

- At key intervals, applications issue a heartbeat (e.g. once per frame)
- Apps also register desired performance (e.g. 30 beats (frames) per second)
- The performance (heart rate) and goals can be read by system software
- If performance is low the system adapts to increase performance
- If performance exceeds goals, the system frees resources

Video Encoder

Goals: 30 beat/s

Self-Aware System

1.6
2.4

Cores

1
16

Speed

1
10

Bandwidth

API

20 b/s
Roles in the SEEC Framework

<table>
<thead>
<tr>
<th>Application Developer</th>
<th>Systems Developer</th>
<th>SEEC System Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Express application goals and progress (e.g. frames/second)</td>
<td></td>
<td>Read goals and performance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Determine how to adapt (e.g. How much to speed up the application)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Provide a set of actions and a callback function (e.g. allocation of cores to process)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initiate actions based on results of decision phase</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 “best” 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.
Each action has the following attributes:

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

- **Cost**
  - Predicted downside of taking an action (increased power, lowered quality)

- **Callback**
  - A function that takes an id and implements the associated action
Decisions are made to select actions given observations:

- Read application goals and heartbeats
- Determine speedup with *adaptive* 2\(^{nd}\) order control system
- Translate speedup into one or more actions

The control system provides predictable and analyzable behavior
Optimizing Resource Allocation with SEEC

- SEEC can observe, decide and act

- How does this enable optimal resource allocation?

- Let’s implement the video encoder example from the introduction
Performance/Watt Adaptation in Video Encoding

Time (s)

Time (Heartbeat)

Performance (Frame/s)

Power (W)

Performance goal

Performance

Power
SEEC’s control system takes actions based on models (of speedup and cost per action) associated with actions. What if the models are inaccurate?
Updating Models in SEEC

- After every action, SEEC updates system models
- Kalman filters used to estimate true speedups

SEEC combines predictability of control systems with adaptability of learning systems
SEEC Online Learning of Speedup Model for Application with Local Minima

- Initial Model
- Actual Speedups
- Learned Speedups
Handling Multiple Applications

- Control actions computed separately for each application
- For finite resources, several alternatives:
  - Priorities determine which apps meet resource needs
  - Weights determine proportional assignment
Outline

• Introduction/Motivating Example

• The SEEC Framework

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>11/13 PARSECs</td>
</tr>
<tr>
<td>Clock Scaler</td>
<td>Change processor speed</td>
<td>Compute vs. Power</td>
<td>11/13 PARSECs</td>
</tr>
<tr>
<td>Bandwidth Allocator</td>
<td>Assign memory controllers to application</td>
<td>Memory vs. Power</td>
<td>STREAM (doesn’t make a difference for 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</td>
<td>Performance vs. Power</td>
<td>PARSECs</td>
</tr>
<tr>
<td></td>
<td>online</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-App Control</td>
<td>Power Manager with multiple applications</td>
<td>Performance vs. Power 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>11/13 PARSECs</td>
</tr>
<tr>
<td>Clock Scaler</td>
<td>Change processor speed</td>
<td>Compute vs. Power</td>
<td>11/13 PARSECs</td>
</tr>
<tr>
<td>Bandwidth Allocator</td>
<td>Assign memory controllers to application</td>
<td>Memory vs. Power</td>
<td>STREAM (doesn’t make a difference for PARSEC)</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 (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>PARSECs</td>
</tr>
<tr>
<td>Multi-App Control</td>
<td>Power Manager with multiple applications</td>
<td>Performance vs. Power for multiple applications</td>
<td>Combinations of PARSECs</td>
</tr>
</tbody>
</table>
Dynamic Knobs: Creating Adaptive Applications

Turn static command line parameters into dynamic structure

**Application Goals**

Maintain performance and minimize quality loss

**System Actions**

Adjust memory locations to change application settings

**Experiment**

Benchmarks: bodytrack, swaptions, SWISH++, x264

Maintain performance when clock speed changes

Detail in Hoffmann et al. “Dynamic Knobs for Power Aware Computing” ASPLOS 2011
Results
Enabling Dynamic Applications

bodytrack

Clock drops 2.4-1.6GHz

Clock rises 1.6-2.4 GHz

w/o SEEC perf. drops

w/ SEEC perf. recovers

SEEC returns quality to baseline
Dynamic knobs automatically enable dynamic response for a range of applications using a single mechanism.

SWISH++

Maintains performance despite noise

swaptions

Perfect behavior

x264

Maintains baseline performance

Results
Enabling Dynamic Applications
## Optimizing Performance per Watt for Video Encoding

Adapt system behavior to needs of individual inputs

### Application Goals
- Maintain 30 frame/s while minimizing power

### System Actions
- Change cores, clock speed, and memory bandwidth

### Experiment
- Benchmark: x264 w/ 16 different 1080p inputs
- Compare performance/Watt w/ SEEC to best static allocation of resources
Results
Optimizing Performance/Watt for Video Encoder

<table>
<thead>
<tr>
<th>Normalized Performance/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>static worst</td>
</tr>
</tbody>
</table>

- blue_sky.yuv
- crowd_run_1080p.yuv
- ducks_take_off_1080p.yuv
- factory.yuv
- in_to_tree_1080p.yuv
- life.yuv
- native_yuv
- park_joy_1080p.yuv
- pedestrian_area_yuv
- riverbed.yuv
- rush_hour.yuv
- station2.yuv
- sunflower.yuv
- tractor.yuv
- average
# Learning Models Online

Adapt system behavior to needs of individual inputs

## Application Goals
- Maintain 30 frame/s while minimizing power

## System Actions
- Change cores, clock speed, and memory bandwidth
- Tailor models to individual applications while running

## Experiment
- Benchmark: x264 w/ 16 different 1080p inputs
- Compare performance/Watt w/ learned model to previous measurements
Results

Performance/Watt with Online Learning

- static worst
- static oracle
- SEEC, known model
- SEEC, learned model

Normalized Performance/Watt

Files:
- blue_sky.yuv
- crowd_run_1080p.yuv
- dinner.yuv
- factory.yuv
- life.yuv
- native.yuv
- park_joy_1080p.yuv
- pedestrian_area.yuv
- riverbed.yuv
- rush_hour.yuv
- station2.yuv
- sunflower.yuv
- tractor.yuv
- average
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 application when clock frequency changes
Results
SEEC Management of Multiple Applications

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
# Summary of Experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Demonstrated Benefit of SEEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Knobs</td>
<td>Maintains application performance in the face of loss of compute resources</td>
</tr>
<tr>
<td>Performance/Watt</td>
<td>Out-performs oracle for static allocation of resources by adapting to fluctuations in input data</td>
</tr>
<tr>
<td>Performance/Watt with learning</td>
<td>Learns models online and still achieve 95% of static oracle</td>
</tr>
<tr>
<td>Multi-App control</td>
<td>Maintains performance of multiple apps by managing algorithm and system resources to adapt to loss of compute resources</td>
</tr>
</tbody>
</table>
Outline

• Introduction/Motivating Example

• The SEEC Framework

• Experimental Validation

Conclusions
SEEC References

• Application Heartbeats framework:

• Control Systems:
  – Maggio, Hoffmann, Santambrogio, Agarwal, Leva. Controlling software applications within the Heartbeats framework. CDC 2010

• Adaptive Applications:

• The SEEC Framework:
Conclusions

• SEEC is designed to help ease programmer burden
  – Solves resource allocation problems
  – Adapts to fluctuations in environment

• SEEC has two distinguishing features
  – Incorporates goals and feedback directly from the application
  – Abstracts sensors, controller, and actuator to create a generic feedback control system capable of managing algorithm, software, and hardware adaptation

• Demonstrated the benefits of SEEC in several experiments
  – SEEC can optimize performance per Watt for video encoding
  – SEEC can adapt algorithms and resource allocation to meet goals in the face of power caps or other changes in environment
Controlling Memory Bandwidth for STREAM

- static min
- static max
- SEEC, pure delay
- SEEC, slow convergence
- SEEC, oscillating
Using Code Perforation to Save Power in Server Farms

• Currently peak load met by provisioning extra hardware

• Instead, we reduce hardware

• At low loads, no perforation necessary

• At high loads, perforation increases capacity
  – Runtime detects performance degradation from load
  – Runtime adjusts perforation in running apps to respond to load
  – Same peak load met with fewer machines

• Tested by consolidating mini-server farm
Power Saving With Code Perforation

- **Power:**
  - Up to 3/4 reduction in machines and power for video
  - Up to 1/3 reduction in machines and power for search
- **Quality:**
  - Max 8% loss for video
  - Max 30% loss for search (Fewer total results – precision of top 10 unchanged)