Dynamic GPGPU Power Management Using Adaptive Model Predictive Control

Abhinandan Majumdar*, Leonardo Piga†, Indrani Paul†, Joseph L. Greathouse†, Wei Huang†, David H. Albonesi*

*Computer Systems Laboratory, Cornell University
†Advanced Micro Devices, Inc.
Model Predictive Control (MPC)

- Previous dynamic power management policies ignore future application behavior
  - Leads to performance loss or wasted energy

- MPC *looks into the future* to determine the best configuration for the current optimization time step

- Though effective in many domains, overheads far too high for short timescales of dynamic power management

- Our approach: An approximation of MPC that dramatically improves GPGPU energy-efficiency with orders of magnitude lower overhead
Dynamic GPGPU Power Management

- Attempts to maximize performance within power constraints

- Hardware knobs
  - Number of active GPU Compute Units
  - DVFS states

- **Goal**: Reduce the energy of the GPU application phases compared to the baseline power manager while matching its performance
Applying MPC to Dynamic Power Management

- General Idea

![Diagram: Horizon H with GPU kernels and apply here mark]
Applying MPC to Dynamic Power Management

- General Idea

[Diagram showing a timeline labeled "Horizon H" with "Apply here" and "GPU kernels" highlighted]
Applying MPC to Dynamic Power Management

- General Idea

- MPC has high overhead
  - Complexity scales exponentially with $H$
  - Minimizing energy under performance cap with discrete HW settings is fundamentally NP-hard
  - Usually requires dedicated optimization solvers, such as CVX, lpsolve, etc.
Approximations to MPC

- Greedy Hill Climbing to reduce the search space

- Static search order heuristic to make MPC optimization polynomial rather than exponential

- Dynamically tuning of the horizon length $H$ to limit the optimization overhead
MPC Power Manager Architecture

- Performance Target
- Performance Model
- Performance Tracker
- MPC Optimizer
- Adaptive Horizon Generator
- Kernel Pattern Extractor
- Performance Counters of Future Kernels
- Optimization Overhead
- Horizon Length
- HW setting
- Power
- Performance
- Performance Feedback
- Max Overhead
- Kernel
MPC Optimizer

- **Objective**: Find minimum energy HW setting for each kernel without impacting overall performance
MPC Optimizer

- **Objective:** Find minimum energy HW setting for each kernel without impacting overall performance

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity
  - Search for the lowest energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determines a static order without backtracking
  - Search cost becomes polynomial
  - Details in the paper

- 65X average reduction in total cost evaluations
Adaptive Horizon Generator

- Longer horizon improves savings but increases MPC optimization time
Adaptive Horizon Generator

- Longer horizon improves savings but increases MPC optimization time

- Limit the overhead to a slowdown factor $\alpha$ by dynamically varying the horizon length

\[ H_i \leq \frac{N}{N} \left( 1 + \alpha - \frac{1}{i} \right) \frac{i \times T_{total}}{N} - \sum_{j=1}^{i-1} (T_j + T_{MPC,j}) \]

\[ \frac{1}{T_{PPK}} \]
Adaptive Horizon Generator

- Longer horizon improves savings but increases MPC optimization time

- Limit the overhead to a slowdown factor $\alpha$ by dynamically varying the horizon length

$$H_i \leq \frac{N}{N} \left(1 + \alpha - \frac{1}{i}\right) \frac{i \times T_{total}}{N} - \sum_{j=1}^{i-1} (T_j + T_{MPC,j})$$

- General Idea:

  - Est. MPC Overhead + Perf. Loss $\leq \alpha$
  - Decrease Horizon Length
  - Increase Horizon Length

No

Yes
MPC in Action – An Example
MPC in Action – An Example

Search order
(2, 1, 4, 5, 3, 6)
MPC in Action – An Example

Past Performance

Future Execution Pattern

Search order
(2, 1, 4, 5, 3, 6)
MPC in Action – An Example

Past Performance

Future Execution Pattern

Search order

(2, 1, 4, 5, 3, 6)
MPC in Action – An Example

Past

Past Performance

Future Execution Pattern

$H = 4$ Time

Optimizer

Perf. Tracker

MPC Optimizer

Search order

$(2, 1, 4, 5, 3, 6)$
MPC in Action – An Example

Past

Setting for kernel 1 applied

Past Performance

Future Execution Pattern

Search order

(2, 1, 4, 5, 3, 6)

Past

Performance

$H = 4$

Time

Optimizer

Perf. Tracker

MPC Optimizer

A

B

$\text{A}$

$\text{B}$

$\text{A}$
MPC in Action – An Example

Past

Updated Past Performance

Search order
(2, 1, 4, 5, 3, 6)

Optimizer

Perf. Tracker

MPC Optimizer

Future Execution Pattern

Past

$H = 5$

Time
MPC in Action – An Example

Past

$H = 5$

Future Execution Pattern

Search order

(2, 1, 4, 5, 3, 6)
MPC in Action – An Example

Past

Setting for kernel 2 applied

Updated Past Performance

Optimizer

Perf. Tracker

MPC Optimizer

$H = 5$

Time

Future Execution Pattern

Search order

(2, 1, 4, 5, 3, 6)
MPC in Action – An Example

Past

Setting for kernel 2 applied

Updated Past Performance

Future Execution Pattern

Search order

(2, 1, 4, 5, 3, 6)
Experimental Testbed

- **AMD A10-7850K APU**
  - 2 out-of-order dual core CPUs
  - GPU contains 512 processing elements (8 CUs) at 720 MHz

- **DVFS states**

<table>
<thead>
<tr>
<th>CPU P States</th>
<th>Voltage (V)</th>
<th>Freq (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1.325</td>
<td>3.9</td>
</tr>
<tr>
<td>P2</td>
<td>1.3125</td>
<td>3.8</td>
</tr>
<tr>
<td>P3</td>
<td>1.2625</td>
<td>3.7</td>
</tr>
<tr>
<td>P4</td>
<td>1.225</td>
<td>3.5</td>
</tr>
<tr>
<td>P5</td>
<td>1.0625</td>
<td>3</td>
</tr>
<tr>
<td>P6</td>
<td>0.975</td>
<td>2.4</td>
</tr>
<tr>
<td>P7</td>
<td>0.8875</td>
<td>1.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NB P States</th>
<th>Freq (GHz)</th>
<th>Memory Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB0</td>
<td>1.8</td>
<td>800</td>
</tr>
<tr>
<td>NB1</td>
<td>1.6</td>
<td>800</td>
</tr>
<tr>
<td>NB2</td>
<td>1.4</td>
<td>800</td>
</tr>
<tr>
<td>NB3</td>
<td>1.1</td>
<td>333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU P States</th>
<th>Voltage (V)</th>
<th>Freq (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM0</td>
<td>0.95</td>
<td>351</td>
</tr>
<tr>
<td>DPM1</td>
<td>1.05</td>
<td>450</td>
</tr>
<tr>
<td>DPM2</td>
<td>1.125</td>
<td>553</td>
</tr>
<tr>
<td>DPM3</td>
<td>1.1875</td>
<td>654</td>
</tr>
<tr>
<td>DPM4</td>
<td>1.225</td>
<td>720</td>
</tr>
</tbody>
</table>

NB and GPU share the same voltage rail

- **Total HW configuration: 336**
Experimental Setup

- 15 GPGPU Benchmarks

- Baseline scheme
  - AMD Turbo Core

- Predict Previous Kernel (PPK)
  - Assume last kernel repeats
  - State-of-the-art: Harmonia ISCA’15, McLaughlin et al. ASBD’14

- Maximum overhead $\alpha = 5\%$

<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmarks</th>
<th>Benchmark Suite</th>
<th>Regular Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>mandelbulbGPU</td>
<td>Phoronix</td>
<td>$A^{20}$</td>
</tr>
<tr>
<td></td>
<td>Nbody</td>
<td>AMD APP SDK</td>
<td>$A^{10}$</td>
</tr>
<tr>
<td></td>
<td>juliaGPU</td>
<td>Phoronix</td>
<td>$A^{10}$</td>
</tr>
<tr>
<td>Irregular with repetitive pattern</td>
<td>EigenValue</td>
<td>AMD APP SDK</td>
<td>$(AB)^{5}$</td>
</tr>
<tr>
<td></td>
<td>XSBench</td>
<td>Exascale</td>
<td>$(ABC)^{2}$</td>
</tr>
<tr>
<td>Irregular with non-repetitive pattern</td>
<td>Spmv</td>
<td>SHOC</td>
<td>$A^{10}B^{10}C^{10}$</td>
</tr>
<tr>
<td></td>
<td>Kmeans</td>
<td>Rodinia</td>
<td>$AB^{20}$</td>
</tr>
<tr>
<td>Irregular with kernels varying with input</td>
<td>swat</td>
<td>OpenDwarfs</td>
<td>Complex pattern</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>Pannotia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pb-bfs</td>
<td>Parboil</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mis</td>
<td>Pannotia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>srad</td>
<td>Rodinia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lulesh</td>
<td>Exascale</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lud</td>
<td>Rodinia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hybridsort</td>
<td>Rodinia</td>
<td></td>
</tr>
</tbody>
</table>
Energy-Performance Gains

![Energy Savings and Relative Performance Graph]

- Energy Savings (%)
- Relative Performance

Predict Previous Kernel
MPC
Energy-Performance Gains

- Energy Savings (%)
- Relative Performance

Predict Previous Kernel
MPC

Energy Savings (%):
- mandelbulbGPU: 20.0%
- NBody: 10.0%
- ibm: 30.0%
- EigenValue: 40.0%
- XSbench: 50.0%
- Spmv: 60.0%
- kmeans: 70.0%
- SWAT: 80.0%
- color: 90.0%
- pb-bfs: 100.0%
- mis: 110.0%
- srad: 120.0%
- lulesh: 130.0%
- lud: 140.0%
- hybridsort: 150.0%
- Irregular-all: 160.0%
- Irregular-var-input: 170.0%
- Average: 180.0%

Relative Performance:
- mandelbulbGPU: 1.1
- NBody: 1.0
- ibm: 0.9
- EigenValue: 0.8
- XSbench: 0.7
- Spmv: 0.6
- kmeans: 0.5
- SWAT: 0.4
- color: 0.3
- pb-bfs: 0.2
- mis: 0.1
- srad: 0.0
- lulesh: -0.1
- lud: -0.2
- hybridsort: -0.3
- Irregular-all: -0.4
- Irregular-var-input: -0.5
- Average: -0.6
Energy-Performance Gains

Energy Savings (%)

- mandelbulpGPU
- NBody
- Ibm
- EigenValue
- XSBench
- Spmv
- kmeans
- swath
- color
- pb-bfs
- mis
- srad
- lulesh
- lud
- hybridsort
- Irregular-all
- Irregular-var-input
- Average

Relative Performance

- mandelbulpGPU
- NBody
- Ibm
- EigenValue
- XSBench
- Spmv
- kmeans
- swath
- color
- pb-bfs
- mis
- srad
- lulesh
- lud
- hybridsort
- Irregular-all
- Irregular-var-input
- Average

Predict Previous Kernel

MPC

24.8%

19.5%

-10%
MPC Overhead

MPC Energy Overhead (%)

MPC Performance Overhead (%)

% Avg. Horizon Length

mandelbulbGPU, NBody, ibm, EigenValue, XSbench, Spmv, kmeans, swat, color, pb-bfs, mis, srad, lulesh, lud, hybridsort, Average
Ramification of Prediction Inaccuracy

- **RF**: 12% Power, 25% Perf
- **15% Power, 10% Perf**: Wu et al. [HPCA 2015]
- **5% Power, 5% Perf**: Paul et al. [ISCA 2015]
Conclusions

- MPC looks into the future to determine the best configuration for the current optimization time step.

- Though effective in many domains, overheads far too high for short timescales of dynamic power management.

- We devise an approximation of MPC that dramatically improves GPGPU energy-efficiency with orders of magnitude lower overhead.

- Our approach reduces energy by 24.8% with only a 1.8% performance impact.
Questions
Backup Slides
Performance-aware Power Management

- Optimum node-level power efficiency is a complex function of SOC configuration, workload characteristics, programming model, and node-level objective/constraints.
Performance-aware Power Management

- Optimum node-level power efficiency is a complex function of SOC configuration, workload characteristics, programming model, and node-level objective/constraints.
Performance-aware Power Management

- Optimum node-level power efficiency is a complex function of SOC configuration, workload characteristics, programming model, and node-level objective/constraints.

HW Knobs: DVFS states of CPU, GPU, NB etc.

Non-linear power vs. performance curves

Time-varying Workload

Normalized kernel throughput

Kernel execution order

0 0.5 1 1.5 2 2.5

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29

mcf hmmer
Performance-aware Power Management

- Optimum node-level power efficiency is a complex function of SOC configuration, workload characteristics, programming model, and node-level objective/constraints.

**HW Knobs:** DVFS states of CPU, GPU, NB etc.

**Non-linear power vs. performance curves**
Performance-aware Power Management

- Optimum node-level power efficiency is a complex function of SOC configuration, workload characteristics, programming model, and node-level objective/constraints.

HW Knobs: DVFS states of CPU, GPU, NB etc.

Non-linear power vs. performance curves

Time-varying Workload

Normalized kernel throughput

Kernel execution order

Maximize Perf under Power/Thermal cap

Minimize Power under Perf cap

Knob 1
High Perf/Watt

Knob 2
Low Perf/Watt

Performance

Watt

Programming Model

Host Tasks
GPU Tasks
User Application
Performance-aware Power Management

- Optimum node-level power efficiency is a complex function of SOC configuration, workload characteristics, programming model, and node-level objective/constraints.

Reduce energy of the GPGPU kernel phase while performing better than a target.
Model Predictive Control (MPC)

- **MPC looks into the future** to determine the best configuration for the current optimization time step.

- Though effective in many domains, overheads far too high for short timescales of dynamic power management.

- **Our approach**: An approximation of MPC that dramatically improves GPGPU energy-efficiency with orders of magnitude lower overhead.
Dynamic GPGPU Power Management

- CPU and GPU consume significant power in servers
- Previous approaches to dynamic power management are locally predictive and ignore future kernel behavior
  - Performance loss or wasted energy
Dynamic GPGPU Power Management

- CPU and GPU consume significant power in servers
- Previous approaches to dynamic power management are locally predictive and ignore future kernel behavior
  - Performance loss or wasted energy

Model Predictive Control

Proactively looks ahead into the future
Dynamic GPGPU Power Management

- CPU and GPU consume significant power in servers
- Previous approaches to dynamic power management are locally predictive and ignore future kernel behavior
  - Performance loss or wasted energy

**Model Predictive Control**

*Proactively* looks ahead into the future

- Applying MPC is challenging for short timescales of dynamic power management
Dynamic GPGPU Power Management

- CPU and GPU consume significant power in servers
- Previous approaches to dynamic power management are locally predictive and ignore future kernel behavior
  - Performance loss or wasted energy

Model Predictive Control
*Proactively* looks ahead into the future

- Applying MPC is challenging for short timescales of dynamic power management
- **Goal:** Approximations to MPC that save GPGPU energy within the timescales of a typical server operation without degrading performance
Dynamic GPGPU Power Management

- **Goal:** Reduce GPGPU energy within the timescales of a typical server operation without degrading performance

- **Computationally Intensive**
  - NP-Hard
  - Challenging for short timescales of dynamic power management

- **Idea:** Improve energy efficiency by looking into future phases
  - Model Predictive Control (MPC)
  - Dynamically vary computation to limit performance overhead
Traditional Energy Management

- **Static**
  - Predefined set of decisions

- **Reactive**
  - Act after sensing a change in behavior

- **Locally Predictive**
  - Predict the immediate behavior
Traditional Energy Management

- **Static**
  - Predefined set of decisions

- **Reactive**
  - Act after sensing a change in behavior

- **Locally Predictive**
  - Predict the immediate behavior

Performance loss or wasted energy
Traditional Energy Management

- **Static**
  - Predefined set of decisions

- **Reactive**
  - Act after sensing a change in behavior

- **Locally Predictive**
  - Predict the immediate behavior

---

**Proactive Energy Management**
Adapt from past and look-ahead into the future
Motivating Future Awareness
Motivating Future Awareness

- **Baseline**
  - AMD Turbo Core

- **Hardware Knobs**
  - DVFS states, GPU CUs
Motivating Future Awareness

- **Baseline**
  - AMD Turbo Core

- **Hardware Knobs**
  - DVFS states, GPU CUs

- **Predict Previous Kernel (PPK):**
  - Assume last kernel repeats
  - State-of-the-art: Harmonia ISCA’15,
    McLaughlin et al. ASBD’14

- **Theoretically Optimal (TO):**
  - Perfect knowledge of future
  - Impractical
Motivating Future Awareness

- **Baseline**
  - AMD Turbo Core

- **Hardware Knobs**
  - DVFS states, GPU CUs

- **Predict Previous Kernel (PPK):**
  - Assume last kernel repeats
  - State-of-the-art: Harmonia ISCA’15, McLaughlin et al. ASBD’14

- **Theoretically Optimal (TO):**
  - Perfect knowledge of future
  - Impractical
Motivating Future Awareness

- **Baseline**
  - AMD Turbo Core

- **Hardware Knobs**
  - DVFS states, GPU CUs

- **Predict Previous Kernel (PPK):**
  - Assume last kernel repeats
  - State-of-the-art: Harmonia ISCA’15, McLaughlin et al. ASBD’14

- **Theoretically Optimal (TO):**
  - Perfect knowledge of future
  - Impractical
Ramifications of Ignoring Future

- Spmv

![Graph showing normalized kernel performance and runtime speedup over kernel execution order for Spmv. The graph compares Baseline, PPK, and Target scenarios.](image-url)
Ramifications of Ignoring Future

- Spmv

![Graph showing normalized kernel performance and runtime speedup over kernel execution order. The graph indicates that PPK lowers performance compared to the baseline and target.](image-url)
Ramifications of Ignoring Future

- Spmv

![Normalized kernel performance graph]

**PPK lowers performance**

- **Baseline**
- **PPK**
- **Target**

**Avoid performance slowdown**

**Look ahead**

![Runtime Speedup graph]

**Baseline**

**PPK**

**Target**

**Performance Loss**
Ramifications of Ignoring Future

- Kmeans
Ramifications of Ignoring Future

- Kmeans

![Graph showing normalized kernel performance, runtime speedup, and target performance with a note that PPK spends a lot of energy.](image-url)
Ramifications of Ignoring Future

- Kmeans

![Graph showing kernel performance, runtime speedup, and energy savings over kernel execution order.](attachment:image.png)

PPK spends lot of energy.
Ramifications of Ignoring Future

- Kmeans

![Graph showing kernel performance, runtime speedup, and energy savings over kernel execution order. The graph illustrates look-ahead strategies to catch up on performance and the energy expenditure of PPK.](image-url)
Background

- Typical GPGPU application phase

![Diagram showing typical GPGPU application phase](image)

- CPU
- DATA TRANSFER
- CPU
- DATA TRANSFER
- CPU
- DATA TRANSFER
- CPU
- GPU Kernel
- DATA TRANSFER
- CPU
- DATA TRANSFER
- CPU
- GPU Kernel
- DATA TRANSFER
- CPU
- DATA TRANSFER
- CPU
- DATA TRANSFER
- CPU

Time
Kernel Performance Scaling

Energy-optimal configuration differ across kernels

- Speedup vs. # of Active GPU CUs for different NBp States
  - (a) Energy-bound: `writeCandidates`
  - (b) Memory-bound: `readGlobalMemoryCoalesced`
  - (c) Peak: `writeCandidates`
  - (d) Unscalable: `astar`
Model Predictive Control (MPC)

- General Idea

Horizon $H$

Apply here
Model Predictive Control (MPC)

- General Idea

![Horizon H]

Apply here
Model Predictive Control (MPC)

- General Idea

- MPC Components
  - Accurate system model
  - Future input forecast
  - Optimization
Model Predictive Control (MPC)

- General Idea

- MPC Components
  - Accurate system model → Power and performance prediction model
  - Future input forecast → Kernel pattern extractor
  - Optimization → Greedy and heuristic based optimizer
Feedback-based Performance Tracker

- Switch the optimization goal based on past performance and the target

- Performance met previously?
  - Yes
    - Optimize Aggressively
      - Reduce energy
      - Avoid performance loss
  - No
    - Relax optimization
      - Spend energy
      - Reduce performance loss
Performance Power Model

- Trained offline using Random Forest Learning Algorithm
- Estimates performance and power for any HW configuration
- Extracts kernel execution pattern upon the first encounter
- Stores the performance counters of dissimilar kernels and retunes it
MPC Optimizer

- Greedy Hill Climbing optimization
  - Using the predictor, select the HW knob with highest energy sensitivity
  - Search for low energy configuration using hill climbing
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Using the predictor, select the HW knob with highest energy sensitivity
  - Search for low energy configuration using hill climbing

CPU DVFS  |  NB DVFS  |  GPU DVFS  |  GPU CU
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Using the predictor, select the HW knob with highest energy sensitivity
  - Search for low energy configuration using hill climbing

![Diagram of GPU DVFS states with Predicted Energy graph]
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Using the predictor, select the HW knob with highest energy sensitivity
  - Search for low energy configuration using hill climbing

```
CPU DVFS  NB DVFS  GPU DVFS  GPU CU
```

![Predicted Energy vs. GPU DVFS states graph]

- Hill climbing
MPC Optimizer

- Greedy Hill Climbing optimization
  - Using the predictor, select the HW knob with highest energy sensitivity
  - Search for low energy configuration using hill climbing

![Graph showing predicted energy vs. GPU CU count with hill climbing at the minimum point.]

- CPU DVFS
- NB DVFS
- GPU DVFS
- GPU CU
MPC Optimizer

- Greedy Hill Climbing optimization
  - Using the predictor, select the HW knob with highest energy sensitivity
  - Search for low energy configuration using hill climbing

![Diagram with predicted energy vs GPU CU count, showing hill climbing and 20x cost reduction over exhaustive search]
MPC Optimizer
MPC Optimizer

- Greedy Hill Climbing optimization
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - General Idea
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - **General Idea**
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - **General Idea**
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - **General Idea**
    - **High to low** performance (e.g. Spmv): Optimize low performing kernels first
    - **Low to high** performance (e.g. Kmeans): Optimize high performing kernels first
  - **Search cost becomes polynomial**

---

**Kernel Performance** vs **Runtime Performance**

- **(High Performance)**
  - Optimize First
- **(Low Performance)**
  - Optimize Second

**Performance normalized to target**

**Kernel Execution Order**

(3, 2, 1, 6, 5, 4)
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - General Idea
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial

65× reduction in cost evaluation
MPC Optimizer
MPC Optimizer

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - **General Idea**
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial
MPC Search Heuristic

- Determine a static order requiring no backtracking
- General Idea
  - High to low performance (e.g. Spmv): Optimize low performing kernels first
  - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
- Search cost becomes polynomial
MPC Search Heuristic

- Determine a static order requiring no backtracking
- General Idea
  - High to low performance (e.g. Spmv): Optimize low performing kernels first
  - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
- Search cost becomes polynomial
MPC Optimizer

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - **General Idea**
    - **High to low performance** (e.g. Spmv): Optimize low performing kernels first
    - **Low to high performance** (e.g. Kmeans): Optimize high performing kernels first
  - **Search cost becomes polynomial**

![Graph showing performance and execution order](image)
MPC Search Heuristic

- Determine a static order requiring no backtracking
- General Idea
  - High to low performance (e.g. Spmv): Optimize **low performing kernels** first
  - Low to high performance (e.g. Kmeans): Optimize **high performing kernels** first
- Search cost becomes polynomial

![Chart](image)

- **(High Performance)** Optimize First
- **(Low Performance)** Optimize Second

65× cost reduction
Adaptive Horizon Generator

MPC runs between kernel invocations
Adaptive Horizon Generator

MPC runs between kernel invocations

\[ M \]
Adaptive Horizon Generator

MPC runs between kernel invocations

- Longer horizon increases MPC optimization time
- Limit the overhead $\alpha$ by dynamically varying the horizon length
Adaptive Horizon Generator

MPC runs between kernel invocations

- Longer horizon increases MPC optimization time
- Limit the overhead $\alpha$ by dynamically varying the horizon length
- General Idea:

$$\frac{(Total\ Time\ of\ i - 1\ kernels) + (Est.\ MPC\ Opt.\ Time) + (Est.\ Time\ of\ kernel\ i)}{Baseline\ Time\ of\ i\ kernels} \leq 1 + \alpha$$
Adaptive Horizon Generator

MPC runs between kernel invocations

- Longer horizon increases MPC optimization time
- Limit the overhead $\alpha$ by dynamically varying the horizon length
- General Idea:

$$H_i \leq \frac{N}{N} \left(1 + \alpha - \frac{1}{i} \right) \frac{i \times T_{total}}{N} - \sum_{j=1}^{i-1} (T_j + T_{MPC,j})$$
Dynamic GPGPU Power Mgmt. Formulation

- Minimize energy over $N$ GPU kernels such that the performance target is met

$$\min_{\vec{s}} \sum_{i=1}^{N} E_i(s_i)$$

such that

$$\frac{\sum_{i=1}^{N} I_i}{\sum_{i=1}^{N} T_i(s_i)} \geq \frac{I_{total}}{T_{total}}$$

$s_i \in S \quad \forall 1 \leq i \leq N$

$S = \overrightarrow{cpu} \times \overrightarrow{nb} \times \overrightarrow{gpu} \times \overrightarrow{cu}$
Theoretically Optimal

- GLPK to solve the Integer Linear Programming (ILP) formulation

\[
\min \sum_{i=1}^{N} \sum_{j \in S} E_i(j)X_{ij}
\]

such that

\[
\sum_{i=1}^{N} I_i - \frac{I_{total}}{T_{total}} \sum_{i=1}^{N} \sum_{j \in S} T_i(j)X_{ij} \geq 0
\]

\[
\sum_{j \in S} X_{ij} = 1 \quad \forall \ 1 \leq i \leq N
\]

\[
X_{ij} \in \{0, 1\} \quad 1 \leq i \leq N \text{ and } \forall j \in S
\]
Predict Previous Kernel (PPK)

- Minimize energy of kernel $i$ such that the runtime performance so far exceeds the target

$$\min_{s_i \in S} E_i(s_i)$$

such that

$$\frac{\sum_{j=1}^{i} I_j}{\sum_{j=1}^{i} T_j(s_j)} \geq \frac{I_{total}}{T_{total}} \quad \forall 1 \leq i \leq N \text{ and } \forall s_j \in S$$
MPC-based GPGPU Power Manager

- Optimize energy for next $H$ kernels such that the runtime performance at the end of $H$ kernels exceeds the target

\[
\min_{\mathcal{S}} \sum_{j=i}^{i+H-1} E_j(s_j)
\]

such that

\[
\frac{\sum_{j=1}^{i+H-1} I_j}{\sum_{j=1}^{i+H-1} T_j(s_j)} \geq \frac{I_{total}}{T_{total}} \quad \forall 1 \leq i \leq N \text{ and } \forall s_j \in S
\]
Runtime Performance Tracker

- Performance requirement of kernel, $k$, is enforced as follows:

$$\frac{\sum_{j=1}^{k-1} I_j + \mathbb{E}[I_k]}{\sum_{j=1}^{k-1} T_j(s_j) + \mathbb{E}[T_k(s_k)]} \geq \frac{I_{total}}{T_{total}}$$

- Kernel time headroom is updated according to:

$$\mathbb{E}[T_k(s_k)] \leq \left(\sum_{j=1}^{k-1} I_j + \mathbb{E}[I_k]\right) \left(\frac{I_{total}}{T_{total}} - \sum_{j=1}^{k-1} T_j(s_j)\right)$$
Feedback-based Performance Tracker

- Performance met previously?
  - Yes: Optimize Aggressively
    - Reduce energy
    - Avoid performance loss
  - No: Relax optimization
    - Spend energy
    - Reduce performance loss
MPC Optimizer
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - General Idea
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial
  - 65X average reduction in total cost evaluations

- An Example
  - Search order = (2, 1, 4, 5, 3)
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - General Idea
    - **High to low performance** (e.g. Spmv): Optimize low performing kernels first
    - **Low to high performance** (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial

- 65X average reduction in total cost evaluations

An Example

- Search order = \((2, 1, 4, 5, 3)\)
MPC Optimizer

- **Greedy Hill Climbing optimization**
  - Select the HW knob with highest energy sensitivity and search for low energy configuration using hill climbing

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - General Idea
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search cost becomes polynomial

- 65X average reduction in total cost evaluations

---

**An Example**

- Search order = \((2, 1, 4, 5, 3)\)
MPC Optimizer

- **Greedy hill climbing optimization**
  - **Greedy**: Select the HW knob with highest energy sensitivity
  - **Hill Climbing**: Continue finding low energy configuration (within perf target), and stop
  - Reduces search cost by 20X

- **MPC Search Heuristic**
  - Determine a static order requiring no backtracking
  - General Idea
    - **High to low** performance (e.g. Spmv): Optimize low performing kernels first
    - **Low to high** performance (e.g. Kmeans): Optimize high performing kernels first
  - Search becomes polynomial

- **An Example**
  - Search order = (2, 1, 4, 5, 3)
MPC Optimizer

- Greedy hill climbing optimization
  - **Greedy**: Select the HW knob with highest energy sensitivity
  - **Hill Climbing**: Continue finding low energy configuration (within perf target), and stop
  - Reduces search cost by 20X

- MPC Search Heuristic
  - Determine a static order requiring no backtracking
  - General Idea
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search becomes polynomial

An Example

- Search order = (2, 1, 4, 5, 3)
MPC Optimizer

- Greedy hill climbing optimization
  - **Greedy**: Select the HW knob with highest energy sensitivity
  - **Hill Climbing**: Continue finding low energy configuration (within perf target), and stop
  - Reduces search cost by 20X

- MPC Search Heuristic
  - Determine a static order requiring no backtracking
  - General Idea
    - High to low performance (e.g. Spmv): Optimize low performing kernels first
    - Low to high performance (e.g. Kmeans): Optimize high performing kernels first
  - Search becomes polynomial
  - Average reduction in total cost evaluations: 65X

An Example

- Search order = (2, 1, 4, 5, 3)
## GPGPU Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmarks</th>
<th>Benchmark Suite</th>
<th>Regular Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>mandelbulbGPU</td>
<td>Phoronix</td>
<td>$A^{20}$</td>
</tr>
<tr>
<td></td>
<td>Nbody</td>
<td>AMD APP SDK</td>
<td>$A^{10}$</td>
</tr>
<tr>
<td></td>
<td>juliaGPU</td>
<td>Phoronix</td>
<td>$A^{10}$</td>
</tr>
<tr>
<td>Irregular with repetitive pattern</td>
<td>EigenValue</td>
<td>AMD APP SDK</td>
<td>(AB)$^5$</td>
</tr>
<tr>
<td></td>
<td>XSBench</td>
<td>Exascale</td>
<td>(ABC)$^2$</td>
</tr>
<tr>
<td>Irregular with non-repetitive pattern</td>
<td>Spmv</td>
<td>SHOC</td>
<td>$A^{10}B^{10}C^{10}$</td>
</tr>
<tr>
<td></td>
<td>Kmeans</td>
<td>Rodinia</td>
<td>AB$^{20}$</td>
</tr>
<tr>
<td>Irregular with kernels varying with input</td>
<td>swat</td>
<td>OpenDwarfs</td>
<td>Complex pattern</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>Pannotia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pb-bfs</td>
<td>Parboil</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mis</td>
<td>Pannotia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>srad</td>
<td>Rodinia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lulesh</td>
<td>Exascale</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lud</td>
<td>Rodinia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hybridsort</td>
<td>Rodinia</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Testbed (Detailed)

- **AMD A10-7850K APU**
  - 2 out-of-order dual core CPUs
  - GPU contains 512 processing elements (8 CUs) at 720 MHz
    - Each CU has 4 SIMD Vector Units
    - 16 PEs per SIMD vector unit

- **DVFS states**
  - NB and GPU share the same voltage rail

<table>
<thead>
<tr>
<th>CPU P States</th>
<th>Voltage (V)</th>
<th>Freq (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1.325</td>
<td>3.9</td>
</tr>
<tr>
<td>P2</td>
<td>1.3125</td>
<td>3.8</td>
</tr>
<tr>
<td>P3</td>
<td>1.2625</td>
<td>3.7</td>
</tr>
<tr>
<td>P4</td>
<td>1.225</td>
<td>3.5</td>
</tr>
<tr>
<td>P5</td>
<td>1.0625</td>
<td>3</td>
</tr>
<tr>
<td>P6</td>
<td>0.975</td>
<td>2.4</td>
</tr>
<tr>
<td>P7</td>
<td>0.8875</td>
<td>1.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NB P States</th>
<th>Freq (GHz)</th>
<th>Memory Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB0</td>
<td>1.8</td>
<td>800</td>
</tr>
<tr>
<td>NB1</td>
<td>1.6</td>
<td>800</td>
</tr>
<tr>
<td>NB2</td>
<td>1.4</td>
<td>800</td>
</tr>
<tr>
<td>NB3</td>
<td>1.1</td>
<td>333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU P States</th>
<th>Voltage (V)</th>
<th>Freq (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM0</td>
<td>0.95</td>
<td>351</td>
</tr>
<tr>
<td>DPM1</td>
<td>1.05</td>
<td>450</td>
</tr>
<tr>
<td>DPM2</td>
<td>1.125</td>
<td>553</td>
</tr>
<tr>
<td>DPM3</td>
<td>1.1875</td>
<td>654</td>
</tr>
<tr>
<td>DPM4</td>
<td>1.225</td>
<td>720</td>
</tr>
</tbody>
</table>

- GPU CUs and DVFS states changed in multiples of 2
- Total HW configuration: 336
Experimental Setup

15 GPGPU Benchmarks

- Sampled from 73 benchmarks
- 75% irregular
- 44% vary with input

Baseline scheme

- AMD Turbo Core

Algorithms

- PPK
- MPC
- TO

Results based on real hardware traces
Polynomial MPC w/ Theoretical Optimal

Energy Savings (%)

MPC
Theoretically Optimal

Speedup

mandelbulpGPU  NBody  lbn  EigenValue  XSbench  Spmv  kmeans  swat  color  pb-bfs  mis  srad  lulesh  lud  hybridsort  Irregular-all  Irregular-var-input  Average
Polynomial MPC w/ Theoretical Optimal

- Energy Savings (%)
- Speedup

Bar chart comparing MPC and Theoretically Optimal for various benchmarks, showing energy savings and speedup percentages. The chart highlights 92% energy savings and 93% speedup on average.
GPU Energy Savings

- Predicted Previous Kernel
- MPC

---

mandelbulp, gpu, NBody, EigenValue, XSBench, Spmv, kmeans, swat, colr, pb-brd, mis, sprd, lulesh, lud, hybridsort, Irregular-all, Irregular-var-input, Average
MPC Energy-Performance w.r.t PPK
Amortization of Initial Losses
Final Takeaway

- MPC is a versatile scheme
  - Does not hurt gains of regular benchmarks
  - Improves energy savings
  - Reduces performance loss

- Online MPC reduces performance loss compared to traditional approaches

- Online MPC is resilient to prediction inaccuracy due to
  - Performance feedback
  - MPC’s greedy and heuristic approximations depend minimally on prediction models