PhD Dissertation Proposal

Characterizing, Optimizing, and Auto-Tuning Applications for Energy Efficiency

Wei Wang

The Committee:
Chair: Dr. John Cavazos
Member: Dr. Guang R. Gao
Member: Dr. James Clause
Member: Dr. Allan Porterfield

January 28, 2015
HPC Energy Optimization Challenge

Table: Performance, power, and energy efficiency of top/green500 and exascale systems

<table>
<thead>
<tr>
<th>System Name</th>
<th>Performance (TFLOP/s)</th>
<th>Power (KW)</th>
<th>GFLOPS/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exascale System</td>
<td>1,000,000</td>
<td>20,000</td>
<td>50</td>
</tr>
<tr>
<td>MilkyWay-2</td>
<td>33,862.7</td>
<td>17,808</td>
<td>1.9</td>
</tr>
<tr>
<td>Titan</td>
<td>17,590.0</td>
<td>8,209</td>
<td>2.14</td>
</tr>
<tr>
<td>L-CSC</td>
<td>301.3</td>
<td>57.15</td>
<td>5.272</td>
</tr>
</tbody>
</table>

Exascale computing requires more than $20 \times$ improvement in GFLOPS/Watts.
Outline

1. Energy Auto-Tuning and Optimization in Polyhedral Optimization Space
2. Energy Optimization with Program Characterization and CPU Clock Modulation
3. Proposed Future Work
Outline

1. Energy Auto-Tuning and Optimization in Polyhedral Optimization Space

2. Energy Optimization with Program Characterization and CPU Clock Modulation

3. Proposed Future Work
Motivation

- Polyhedral optimization effective for optimizing computational kernels
- Accurate predictive performance model derived
- Effective Predictive Performance Model $\iff$ Effective Predictive Energy Model?
Adapting AutoTuning Framework for Energy

- Program Characterization
  - Loop Pattern
  - Control Flow Graph
- Optimization Sequences
  - Src-to-Src Compiler
- Energy Profiling
  - Energy Counters
- Machine Learning Algorithms
  - SVM

Figure: Auto-Tuning framework for energy. (Refs: Park et al. CGO’11, CGO’12, IJPP’13, ICPP’14)
Energy Measurement using RCRTool

**SandyBridge**
- Monitors MSR Energy Counter
- 1000Hz+ update frequency
- Measures energy, computes power

**KNIGHT's Corner**
- Built-in power measurement used (/sys/class/micras/power)
- 20Hz update frequency
- Measures power, computes energy

**Figure**: Simplified view of RCRtool energy monitoring
Energy Measurement APIs

```c
int main() {
    initialize();

    while {
        #pragma omp parallel for
        compute_region_1();

        serial_code();

        #pragma omp parallel
        #pragma for
        compute_region_2();

    }

    finalize();
}
```

**Figure:** Original Program

```c
int main() {
    energyDaemonInit();
    initialize();

    energyDaemonTEnStart();
    #pragma omp parallel for
    compute_region_1();

    energyDaemonEnter("thefile", 10);

    serial_code();

    energyDaemonEnter();
    #pragma omp parallel
    #pragma for
    compute_region_2();

    energyDaemonExit("the file", 18);

    energyDaemonTEnStop();
    finalize();
    energyDameonTerm();
}
```

**Figure:** Added with energy profiling APIs
Polyhedral Compilers

Generate code variants of programs containing Static Control Parts (SCoP) using PoCC (Polyhedral Compiler Collection)

- Loop Transformations
- Auto Parallelization (PLUTO)
- Tested Applications
  - Existing: Polybench
  - New: 2D Cardiac Wave Propagation Simulation, LULESH
Exposing SCoP

```c
/** Original Version ******************/
double abfun(double a) {
  return exp(a+0.1)*0.2;
}

/** Transformed Version ******************/
step = 0;
while (step < T) {
  if (step % 2 == 0) {
    for (i=0; i<M; i++) {
      for (j=0; j<N; j++) {
        double temp0,temp1,temp2,v;
        temp0 = abfun(a[step%2][0][i][j]);
        temp1 = abfun(a[step%2][1][i][j]);
        temp2 = abfun(a[step%2][2][i][j]);
        v = (temp0 + temp1) / temp2;
      }
    }
    step++;
  } else {
    for (i=0; i<M; i++) {
      for (j=0; j<N; j++) {
        double v;
        v = (exp(a[1][0][i][j]+0.1)*0.2 + exp(a[1][1][i][j]+0.1)*0.2) /
            exp(a[1][2][i][j]+0.1)*0.2;
      }
    }
    step++;
  }
}
```

**Figure:** Simplified version of the original and the transformed loop nest
Energy Profiling of Different Program Optimizations

Figure: Workflow of energy-aware polyhedral framework
Experiments Setup

- **Hardware**
  - Intel Xeon E5-2680 (dual socket 8-core processor with 20MB cache)
  - Xeon Phi coprocessor (61 cores, 1.09GHz, 512KB cache each)

- **Software**
  - Polyhedral Compilers: PoCC v1.2 and Polyopt v0.2.1
  - Application: Polybench v3.2 and LULESH v1.0 (OpenMP)
  - Back-end Compilers: GCC v4.4.6 and ICC v14.0.0
Energy Consumption and Execution Time Correlation (Polybench)

Best optimizations for time are best for energy savings for these two polybench application.
Jumps in Seidel2D energy usage (and decreased execution time) are results of turning parallelization on.
Polyhedral Optimizations on a Realistic Application

2D Cardiac Wave Propagation Simulation

Energy/Performance improvement on the Sandy Bridge system

Baseline: manual OpenMP implementation
Conclusion: Polyhedral approach is effective in optimizing the 2D Cardiac Wave Propagation Simulation.
Energy Consumption and Execution Time Correlation (2D Cardiac Wave Propagation Simulation)

<table>
<thead>
<tr>
<th></th>
<th>Energy (joules)</th>
<th>Execution Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>10500</td>
<td>10</td>
</tr>
<tr>
<td>6500</td>
<td>10000</td>
<td>15</td>
</tr>
<tr>
<td>7000</td>
<td>9500</td>
<td>20</td>
</tr>
<tr>
<td>7500</td>
<td>9000</td>
<td>25</td>
</tr>
<tr>
<td>8000</td>
<td>8500</td>
<td>30</td>
</tr>
<tr>
<td>8500</td>
<td>8000</td>
<td>35</td>
</tr>
<tr>
<td>9000</td>
<td>7500</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Energy (joules)</th>
<th>Execution Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3000</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>4000</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>5000</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>6000</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>7000</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>8000</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

Left: Sandy Bridge  
Right: Xeon Phi  
Conclusion: Saving energy consumption is consistent with improving performance on both processors
Conclusion

- Tuning for time can be used as proxy to tuning for energy
- Polyhedral optimizations for realistic applications possible
Outline

1. Energy Auto-Tuning and Optimization in Polyhedral Optimization Space
2. Energy Optimization with Program Characterization and CPU Clock Modulation
3. Proposed Future Work
HPC energy optimizations focus on DVFS
DVFS only applied in the coarse-grain cases
Fine-grained energy control requires faster frequency transition techniques
CPU Clock Modulation

- Write Specific Value to IA32_CLOCK_MODULATION (0x19a) MSR
- Modify /dev/cpu/cpu{0:15}/msr with root privilege
- Invoke wrmsr inline assembly from applications using added System Call

# Available Frequencies

<table>
<thead>
<tr>
<th>Duty Cycle Level</th>
<th>Binary</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Effective Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10001B</td>
<td>17</td>
<td>11H</td>
<td>6.25%</td>
</tr>
<tr>
<td>2</td>
<td>10010B</td>
<td>18</td>
<td>12H</td>
<td>12.5%</td>
</tr>
<tr>
<td>3</td>
<td>10011B</td>
<td>19</td>
<td>13H</td>
<td>18.75%</td>
</tr>
<tr>
<td>4</td>
<td>10100B</td>
<td>20</td>
<td>14H</td>
<td>25%</td>
</tr>
<tr>
<td>5</td>
<td>10101B</td>
<td>21</td>
<td>15H</td>
<td>31.25%</td>
</tr>
<tr>
<td>6</td>
<td>10110B</td>
<td>22</td>
<td>16H</td>
<td>37.5%</td>
</tr>
<tr>
<td>7</td>
<td>10011B</td>
<td>23</td>
<td>17H</td>
<td>43.75%</td>
</tr>
<tr>
<td>8</td>
<td>11000B</td>
<td>24</td>
<td>18H</td>
<td>50%</td>
</tr>
<tr>
<td>9</td>
<td>11001B</td>
<td>25</td>
<td>19H</td>
<td>56.25%</td>
</tr>
<tr>
<td>10</td>
<td>11010B</td>
<td>26</td>
<td>1AH</td>
<td>63.5%</td>
</tr>
<tr>
<td>11</td>
<td>11011B</td>
<td>27</td>
<td>1BH</td>
<td>69.75%</td>
</tr>
<tr>
<td>12</td>
<td>11100B</td>
<td>28</td>
<td>1CH</td>
<td>75%</td>
</tr>
<tr>
<td>13</td>
<td>11101B</td>
<td>29</td>
<td>1DH</td>
<td>81.25%</td>
</tr>
<tr>
<td>14</td>
<td>11110B</td>
<td>30</td>
<td>1EH</td>
<td>87.5%</td>
</tr>
<tr>
<td>15</td>
<td>11111B</td>
<td>31</td>
<td>1FH</td>
<td>93.75%</td>
</tr>
<tr>
<td>16</td>
<td>00000B</td>
<td>0</td>
<td>00H</td>
<td>100%</td>
</tr>
</tbody>
</table>
Benchmarks and Experimental Setup

1 Benchmarks
   - LULESH: Hydrodynamics
   - miniFE from Mantevo Project: implicit finite-element application
   - brdr2d: 2D Cardiac Wave Propagation Simulation
   - Polybench: 30 Computational Kernels

2 Hardware/Software Setup
   - Intel Xeon E5-2680 (Dual Socket, 8-core processor with 20MB LLC, 2.7GHz)
   - Linux 2.6.32 with ACPI and MSR modules
   - Intel ICC v14.0.2 with -O3
   - RCRdaemon taskset to core 16
Loops with High Memory Access

(a) LULESH with DCM

(b) jacobi-2d Polybench
Loops with High Memory Access

Energy reduced and EDP lowered with very low performance impact

(a) LULESH with DCM

(b) jacobi-2d Polybench
Loops with Low Memory Access (More Computation)

(a) LULESH

(b) miniFE
Loops with Low Memory Access (More Computation)

Energy reduced with high performance impact, resulting in higher EDP

(a) LULESH

(b) miniFE
Loops with Balanced Memory Access and Computation

Normalized Metrics
- Duty Cycle (Clock Skipping)
- Energy
- Time
- EDP

(a) LULESH

(b) miniFE
Energy reduced and EDP lowered with relatively low performance impact

(a) LULESH

(b) miniFE
Benchmarks

Polybench programs running at the best non-full speed setting
Loops have different energy characteristics responding to frequency changes.
Memory Access Density vs. Three Types of Loops

Normalized EDP

EDP  MAD

Benchmarks

durbin  adi  jacobi-2d-imper  fddt-2d  lu  cholesky  floyd-warshall  gesummv  covariance  gram-schmidt  ludcmp  dynprog  2mm  3mm  syrk  hmm  correlation  dotgen  fddt-apml  trisolv  atax  bicg  seidel-dt  reg-det  2k  sty/ihm

0.7  0.75  0.8  0.85  0.9  0.95  1  1.05  1.1

0  10  20  30  40  50

MAD Value
Memory Access Density could be used as loop type indicator.
## Loop Characterization Summary

<table>
<thead>
<tr>
<th>Loop Type</th>
<th>Energy Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem-Int.</td>
<td>Power reduced and EDP significantly lowered.</td>
</tr>
<tr>
<td>Comp-Int.</td>
<td>Full frequency required</td>
</tr>
<tr>
<td>Balanced</td>
<td>Saves energy with relatively low performance degradation.</td>
</tr>
</tbody>
</table>
Hybrid Execution of Multi-loop Applications

- Adding energy control APIs around loops
- Fine-grain loop regions require fast machine power-state transition to avoid overhead

```c
while (condition) {
    ...
    setLowFrequency();
    for (i=0; i<N; i++) {
        ...
    }
    resetFrequency();
    for (j=0; j<N; j++) {
        ...
    }
}
```
Table: Comparison of execution time, energy consumption, and EDP for LULESH

<table>
<thead>
<tr>
<th>Version</th>
<th>Duty Cycle Level</th>
<th>Time</th>
<th>Energy</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>minT</td>
<td>100%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>minE</td>
<td>56.25%</td>
<td>1.542</td>
<td>0.743</td>
<td>1.145</td>
</tr>
<tr>
<td>minEDP</td>
<td>81.25%</td>
<td>1.157</td>
<td>0.816</td>
<td>0.943</td>
</tr>
<tr>
<td>Hybrid 1</td>
<td>100% &amp; 50%</td>
<td>1.049</td>
<td>0.882</td>
<td>0.925</td>
</tr>
<tr>
<td>Hybrid 2</td>
<td>100% &amp; 62.5%</td>
<td>1.020</td>
<td>0.897</td>
<td>0.914</td>
</tr>
<tr>
<td>Hybrid 3</td>
<td>100% &amp; 68.75%</td>
<td>1.015</td>
<td>0.914</td>
<td>0.928</td>
</tr>
</tbody>
</table>
miniFE Results

**Table:** Comparison of execution time, energy consumption, and EDP for miniFE

<table>
<thead>
<tr>
<th>Version</th>
<th>Duty Cycle Level</th>
<th>Time</th>
<th>Energy</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>minT</td>
<td>100%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>minE</td>
<td>62.5%</td>
<td>1.351</td>
<td>0.763</td>
<td>1.031</td>
</tr>
<tr>
<td>minEDP</td>
<td>81.25%</td>
<td>1.153</td>
<td>0.819</td>
<td>0.945</td>
</tr>
<tr>
<td>Hybrid 1</td>
<td>100% &amp; 81.25%</td>
<td>1.029</td>
<td>0.893</td>
<td>0.919</td>
</tr>
<tr>
<td>Hybrid 2</td>
<td>100% &amp; 87.5%</td>
<td>1.023</td>
<td>0.923</td>
<td>0.944</td>
</tr>
<tr>
<td>Hybrid 3</td>
<td>100% &amp; 93.75%</td>
<td>1.000</td>
<td>0.954</td>
<td>0.954</td>
</tr>
</tbody>
</table>
Fine-grained DVFS vs. Fine-grained DCM (Entire Application)

<table>
<thead>
<tr>
<th>Hybrid Versions</th>
<th>Time</th>
<th>Energy</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-DCM-50%</td>
<td>2.7G &amp; 1.35G</td>
<td>1.049</td>
<td>0.882</td>
</tr>
<tr>
<td>H-DCM-62.5%</td>
<td>2.7G &amp; 1.69G</td>
<td>1.020</td>
<td>0.897</td>
</tr>
<tr>
<td>H-DCM-68.75%</td>
<td>2.7G &amp; 1.86G</td>
<td>1.015</td>
<td>0.914</td>
</tr>
<tr>
<td>Hybrid-DVFS-2.6G</td>
<td>2.7G &amp; 2.6G</td>
<td>1.004</td>
<td>0.967</td>
</tr>
<tr>
<td>Hybrid-DVFS-2.5G</td>
<td>2.7G &amp; 2.5G</td>
<td>1.051</td>
<td>0.965</td>
</tr>
<tr>
<td>Hybrid-DVFS-2.4G</td>
<td>2.7G &amp; 2.4G</td>
<td>1.086</td>
<td>0.958</td>
</tr>
<tr>
<td>Hybrid-DVFS-2.3G</td>
<td>2.7G &amp; 2.3G</td>
<td>1.122</td>
<td>0.943</td>
</tr>
<tr>
<td>Hybrid-DVFS-2.2G</td>
<td>2.7G &amp; 2.2G</td>
<td>1.129</td>
<td>0.904</td>
</tr>
<tr>
<td>Hybrid-DVFS-2.1G</td>
<td>2.7G &amp; 2.1G</td>
<td>1.186</td>
<td>0.907</td>
</tr>
<tr>
<td>Hybrid-DVFS-1.8G</td>
<td>2.7G &amp; 1.8G</td>
<td>1.333</td>
<td>0.879</td>
</tr>
</tbody>
</table>
Measure and compare the execution time and power of Loop1 and Loop2 with and without energy control APIs.

```c
while (condition) {
    ...
    //Loop1
    MemLoop();
    //Loop2
    CompLoop();
    OtherLoops();
}
```

VS.

```c
while (condition) {
    ...
    setFrequency();
    MemLoop();
    resetFrequency();
    CompLoop();
    OtherLoops();
}
```
Power Transition Overhead

Normalized Metric
Loop 1 (MemLoop)

Normalized Metric
Loop 2 (CompLoop)

DVFS-2.4GHz
DVFS-2.1GHz
DVFS-1.8GHz
DCM-68.75%
DCM-62.5%
DCM-50%

Time
Power

0 0.5 1 1.5

0 0.5 1 1.5
DVFS energy control is not synchronized with fine-grain loops, but DCM is.
Fine-Grain vs. Coarse-Grain DVFS/DCM Energy Control of Loop 1 (MemLoop)

Energy Control Settings

- DCM-50%
- DCM-62.5%
- DCM-68.75%
- DVFS-1.8GHz
- DVFS-2.1GHz
- DVFS-2.4GHz

Normalized Metric

FG-Time
FG-Power
CG-Time
CG-Power
Fine-Grain vs. Coarse-Grain DVFS/DCM Energy Control of Loop 1 (MemLoop)

DCM fine-grain energy control is almost identical to coarse-grain.

DVFS overhead is larger than DCM.
Clock Skipping with Concurrency Throttling

- Concurrency Throttling mitigates resource contention
- Clock Modulation reduces idle state power

(c) Energy with concurrency throttling and clock skipping. Minimum occurs at (75%, 4)

(d) Time with concurrency throttling and clock skipping. Minimum occurs at (100%, 6)

Figure: fdtd-2d Polybench
(a) Energy results. Minimum occurs at (75%, 6)

(b) Time results. Minimum occurs at (100%, 8)

<table>
<thead>
<tr>
<th>Version</th>
<th># of Threads</th>
<th>Duty Cycle Level</th>
<th>Time</th>
<th>Energy</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>16</td>
<td>100%</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CT</td>
<td>6</td>
<td>100%</td>
<td>0.92</td>
<td>0.94</td>
<td>0.87</td>
</tr>
<tr>
<td>CT+CS</td>
<td>6</td>
<td>75%</td>
<td>0.95</td>
<td>0.87</td>
<td>0.83</td>
</tr>
</tbody>
</table>
(a) Energy results. Minimum occurs at (75%, 8)

(b) Time results. Minimum occurs at (100%, 14)

<table>
<thead>
<tr>
<th>Version</th>
<th># of Threads</th>
<th>Duty Cycle Level</th>
<th>Time</th>
<th>Energy</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>16</td>
<td>100%</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CT</td>
<td>10</td>
<td>100%</td>
<td>1.02</td>
<td>0.86</td>
<td>0.88</td>
</tr>
<tr>
<td>CT+CS</td>
<td>10</td>
<td>81.25%</td>
<td>1.06</td>
<td>0.79</td>
<td>0.84</td>
</tr>
</tbody>
</table>
When Concurrency Throttling is Not Beneficial

(a) Energy results. Minimum occurs at (75%, 16)

(b) Time results. Minimum occurs at (100%, 16)

Figure: brdr2d results when applying both concurrency throttling and clock skipping.
Concurrency Throttling and Memory Access Density

Normalized Time/EDP

Number of Threads

Benchmarks

Durbin
Adi
Jacobi-2D imper
FDTD-2D
LU
Jacobi-1D imper
Cholesky
Floyd-Warshall
Gesummv
Covariance
Gemm
Golschmidt
Ludcmp
Dynprog
2Hm
Syrk
1Hm
Correlation
Fortgen
Fidd Apnl
Thissolv
2k
Ax
Bicg
Seidel Dect
Reg Dect
Sy
Smm
Loops with high Mem-value tend to benefit from concurrency throttling.
Hybrid execution of OpenMP loops with Clock Modulation can achieve better energy efficiency

Concurrency throttling can be combined with Clock Modulation to save more energy
Outline

1. Energy Auto-Tuning and Optimization in Polyhedral Optimization Space
2. Energy Optimization with Program Characterization and CPU Clock Modulation
3. Proposed Future Work
Future Work

2. Enhance results with frequency/threads configuration (Aug. 2015)
### Future Work

#### Power Impacts of Polyhedral Optimizations (Dec. 2015)

<table>
<thead>
<tr>
<th>Program Variants</th>
<th>With-tiling-Power</th>
<th>W/o-tiling-Power</th>
<th>With-tiling-Time</th>
<th>W/o-tiling-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power (Watts)</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
<tr>
<td>Execution Time (seconds)</td>
<td>45</td>
<td>50</td>
<td>55</td>
<td>60</td>
</tr>
</tbody>
</table>

**Figure:** Covariance Polybench program variants with and without loop tiling

#### Extending DCM Energy Optimization Technique (Dec. 2015)

- MPI/OpenMP programs
- Runtime Control
- Automating frequency/number-of-threads configuration