Energy Tuning using Polyhedral Approach

Wei Wang[1], John Cavazos[1], Allan Porterfield[2]
[1] Dept. of Computer and Info. Sciences, University of Delaware
[2] RENCI, University of North Carolina-Chapel Hill

1. Introduction

As the HPC community moves into the exascale computing era, application energy has become a big concern. Tuning for energy will be essential in the effort to overcome the limited power envelope. How is tuning for lower energy related to tuning for faster execution? In this presentation, a strong correlation is presented between the two that allows tuning for execution to be used as a proxy for energy tuning. We also show that polyhedral compilers can effectively tune a realistic application for both time and energy on different architectures including Intel Many Integrated Core (MIC) architecture.

2. Tools

- Energy Measurement
  - RCRTool: Monitors Energy Consumption
    - On Sandy Bridge: Intel RAPL(Running Average Power Limit) Energy Counters, update freq: > 1000/s
    - On Xeon Phi: /sys/class/micras/power, update freq: ~50/s
    - Unified API: energyStatStart() & energyStatEnd()
- Polyhedral Optimization
  - PoCC: Polyhedral Compiler Collection
  - PolyOpt: Polyhedral Optimizer for ROSE Compiler

3. Polyhedral Framework

This work extends previous work to study the polyhedral framework's energy impact.

Four Steps:

1. Loop Transformation
2. Add APIs using ROSE
3. Compilation
4. Execution

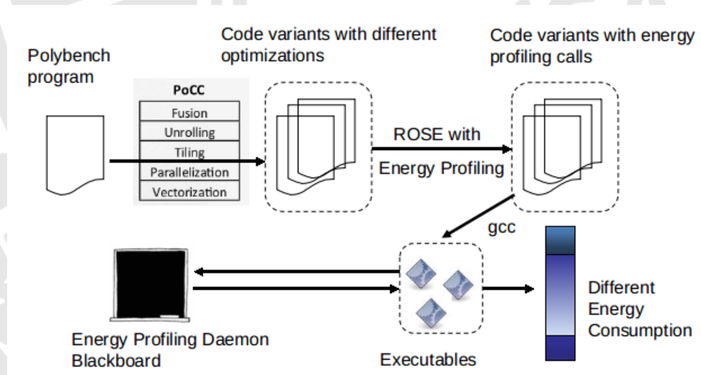

Figure 1: Energy-aware Polyhedral Framework

4. Benchmarks

- Polybench
  - 30 Programs containing small kernels
  - Program variants generated by PoCC
- LULESH
  - Shock hydrodynamic simulation
  - Program variants generated by PolyOpt
- A Realistic Application
  - 2D cardiac wave propagation simulation
  - Program variants generated by PolyOpt

5. Results

1. Energy usage of (and the execution time) of the Polybench programs and LULESH on the Intel Sandy Bridge architecture.
2. A realistic application on both the Intel Sandy Bridge and the Intel Xeon Phi architectures.

5.1 Execution Time and Energy Correlation

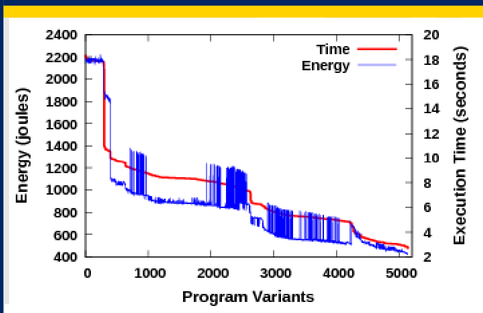

Figure 2: Graph showing the relationship between the execution time and the energy consumption of all covariance Polybench program variants on Sandy Bridge Processor (sorted by execution time). The spikes are caused by "maxfuse" loop transformation.

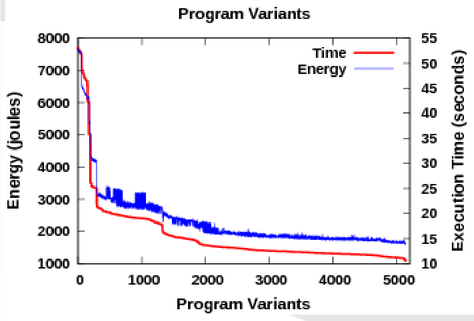

Figure 3: Graph showing the correlation between the execution time and the energy consumption of 2mm Polybench on Sandy Bridge Processor (sorted by execution time). The spikes are caused by bad tiling configuration.

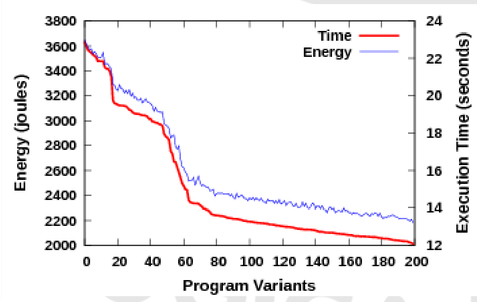

Figure 4: Graph showing the correlation between the execution time and the energy consumption of LULESH program on Sandy Bridge Processor (sorted by execution time).

5.2 Polyhedral Optimization on Realistic App.

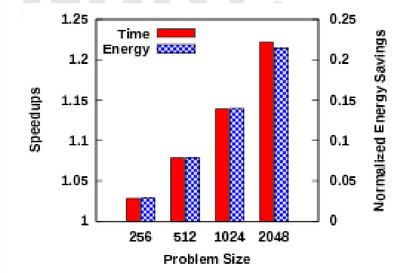

Figure 5: Graph showing the performance improvement and energy savings of the optimal program variant over the baseline OpenMP implementation for different problem size on Sandy Bridge Processor.

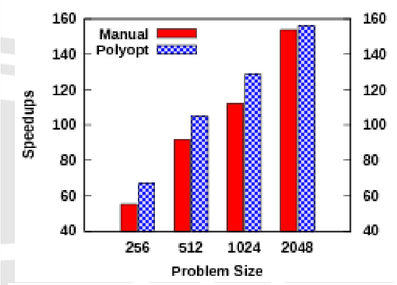

Figure 6: Graph showing the comparison between speedups of manual OpenMP implementation and the best Polyopt/PoCC generated OpenMP program variant over the sequential implementation on MIC architecture.

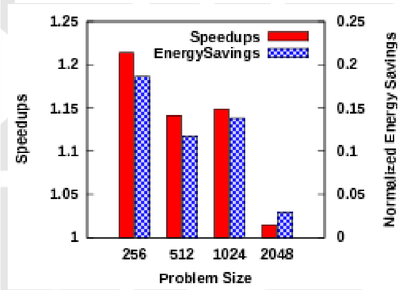

Figure 7: Graph showing the performance improvement and energy savings of the optimal Polyopt/ PoCC generated program variant over the baseline OpenMP implementation for different problem size on MIC architecture.

6. Conclusion

- Optimizations can increase the power and energy required between variants, but variant with minimum execution time also has the lowest energy usage.
- On different architectures, improvements as high as 20% in execution time and a similar amount of reduction in energy (for a realistic application) are obtained using polyhedral approach.