Compilation Workflow

1. Original Polybench program with no modifications
2. Change generic Polybench header file to include energy monitoring API (RCRtool)
3. Use PoCC to generate space-time versions of original polybench program
4. Use Intel® Compiler to compile the version codes targeting Knight’s Corner or SandyBridge
5. Execute all binaries and manually analyze energy/performance results

Results

- Generated 2535 versions of each benchmark with two different dataset sizes (Small and Large)
- CPU configuration: dual-socket quad-core Intel® Xeon® CPU ES-2603 v1.80GHz, 32GB DDR3
- MIC configuration: 60-core Intel® Xeon® Phi Coprocessor S110P @ 1.053 GHz, 8GB DDR5
- Compilation configuration: CPU: -O3 -xHOST -openmp MIC: -O3 -mic -openmp
- Ran each benchmark 5 times and took the average of the middle three executions

Energy Tuning of Polyhedral Kernels on Multicore and Many-Core Architectures

William Killian, Wei Wang, Eunjung Park, John Cavazos
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE, UNIVERSITY OF DELAWARE

Introduction

- Reducing application energy consumption is important in embedded systems.
- Applications are not energy-aware in its design stage.
- Tuning for energy efficiency is necessary to meet system energy constraints.

METHODOLOGY:

- We tuned for energy consumption on Intel® SandyBridge and Intel® Knight’s Corner.
- We applied polyhedral optimizations to kernels
- We found the optimization sequence yielding minimal energy consumption.

EVALUATION:

- In-depth comparison on SandyBridge of execution time and energy consumption
- Cross-architecture comparison of optimization sequences

Energy Measurement

- Used RCRtool [3] to report region-based energy consumption
- Recorded elapsed time, energy use, and average power for region + application

SANDYBRIDGE:

- RAPL hardware counter used
- 1000Hz+ update frequency
- Measures energy, computes power

KNIGHT’S CORNER:

- Built-in power measurement used (/sys/class/powercap)
- 20Hz update frequency
- Measures power, computes energy

Polyhedral Optimizations

- Use the polyhedral model of an AST to transform loops
- Examples include loop unrolling, tiling, fusion, autoparallelization, and auto-vectorization
- Used the Polyhedral Compiler Collection (PoCC) [2] for automatic version generation

LOOP UNROLLING:

```c
// nu unrolling
for (int i = 0; i < n; i++) {
    C[i] = A[i] + B[i];
}
```

SAMPLE USAGE OF POCC:

```c
5 pocc --pluto-parallel --pragma-mark-par-loops [-pluto-prevector] [-pluto-tile N x N] [-pluto-fuse (maxfuse, nofuse, smartfuse)]
```

Optimization Flag Affecting:

- LOOP FUSION
- LOOP UNROLLING
- LOOP TILING
- AUTOPARALLELIZATION
- AUTOVECTORIZATION

TILE SIZES: 1,1632,64 UNROLL FACTORS: 2,4,6,8

Note: we always selected an optimization sequence with auto-parallelization. All other optimizations were optional.

Benchmarks

- Synthetic benchmark is a subset of the Polybench kernels [1] ranging from various domains

POLYBENCH KERNELS:

- 2mm – two matrix multiplication (C = A*B, E = C*D)
- covariance – covariance computation (data mining)
- gemm – matrix multiplication (C = alpha*A*B + beta*C)
- gramschmidt – decomposition (linear algebra)
- jacobi-2d – 2D Jacobi stencil (fluid dynamics, image processing)
- seidel-2d – 2D Seidel stencil computation

Cost Compared to Optimal

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>2mm</th>
<th>covariance</th>
<th>gemm</th>
<th>gramschmidt</th>
<th>jacobi-2d</th>
<th>seidel-2d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost Compared</td>
<td>100%</td>
<td>150%</td>
<td>200%</td>
<td>250%</td>
<td>300%</td>
<td>350%</td>
</tr>
<tr>
<td>Speedup</td>
<td>1.48x</td>
<td>1.36x</td>
<td>1.28x</td>
<td>0.77x</td>
<td>0.78x</td>
<td>0.57x</td>
</tr>
<tr>
<td>Energy Reduction Factor</td>
<td>200%</td>
<td>300%</td>
<td>400%</td>
<td>700%</td>
<td>50%</td>
<td>0%</td>
</tr>
</tbody>
</table>

These graphs indicate that good loop transformations for one architecture do not carry over to other architectures. Performance improvements at times approached or even beat the best optimization sequence. When analyzing total energy consumption, this trend no longer holds.

Conclusion

- Polyhedral optimizations yield up to 60x energy reduction and 120x speedup (28% minimum)
- Non-correlated speedup observed on dense matrix kernels
- Loop transformations do not carry over well across architectures
- Adapting energy-aware API to new benchmarks is easy with RCRTool

FUTURE WORK:

- Explore new benchmark suites and architectures (ARM, Haswell, Silvermont)
- Expand search-space to include more polyhedral optimizations; reduce search breadth
- Improve energy measurement granularity on Knight’s Corner

SOURCES / RELEVANT WORK: