Power Impact of Polyhedral Optimzations on Multicore Architecture

I. Introduction

Previously we have observed that tuning for execution can be used as a proxy to tuning for EDP (Energy-Delay Product). In this report, we look deeper into the power impact of individual loop optimizations and the interactive effect on power and execution time when multiple loop optimizations are applied. This report shed light on how, when and why do optimizations interact.

II. Benchmarks and Experimental Setup

In this work, we evaluate `covariance` Polybench program for understanding the power impact of different loop optimizations. The tests ran on a 2-socket 8-core Intel Xeon E5-2680 processor with 20MB (40MB total) L3 cache. PoCC v1.2 was used to generate program variants from Polybench v3.2. The extra large data set (specified in Polybench) was used. GCC v4.4.6 was the backend compiler. Every executable was compiled with -O3 optimization flag.

III. Experimental Results

Fig. 1 compares the execution time and the power consumption of `covariance` program variants with and without a good tiling size configuration ($32 \times 16 \times 1$). We can see from comparing the pink/purple line with the green line that proper tile size reduces execution time by up to $4\times$ (always at least $2\times$ performance increase) and results in minimum execution time.

We can also see from the smooth execution time line that with tiling there are 2 performance breaks and without tiling 1 performance break. The fastest executions of variants with and without tiling all have maxfuse set – approx 45% improvement of performance. In the case of program variants with tiling,

1) 1st break. Turning vectorization OFF had 45% improvement
2) 2nd break. Setting maxfuse on had 45% improvement

Optimizations can greatly change the power required by an application. E.g. maxfuse takes the tiled version from 105W to 150W (40% increase), the untiled from 120W to 170W (again about 40% increase). Also note that turning vectorization off increased power by a couple of Watts. For this application the power increases

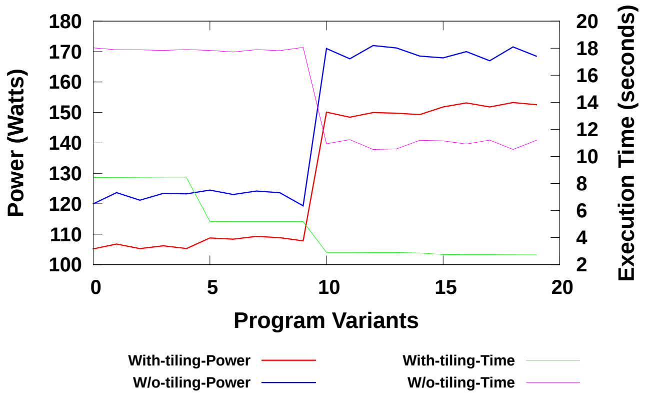


Fig. 1. Graph showing the execution time and the power of `covariance` Polybench program variants with and without loop tiling on SandyBridge Processor (sorted by execution time of program variants with loop tiling turned on). The jumps of power are caused by "maxfuse" loop transformation. The two breaks of the execution time line (with loop tiling) are caused by turning vectorization off and turning maxfuse on, respectively.

are more than made up by the execution time reductions – least energy is the quickest execution time.

IV. Related Work

Is energy-minimum = time minimum?

Yes. [**?**] applied polyhedral optimizations to generate multiple program variants with various optimizations including loop fusion, unrolloing, tiling, and vectorization for OpenMP programs. Their results show a strong correlation between execution time and energy consumption.

V. Todo

Evaluate the effect of putting a power cap on the system.

A. *Negative/Not Exciting Results*

Approach: we developed a unbalanced OpenMP program. The workload of a thread is a linear function of its thread id. Using gcc (v4.8.1). We manually set the duty cycle (slowing down the threads that have fewer amount of workload and let the thread run at full speed if the workload for it is high). Then we set the power cap on the system. Figure **??** and Figure 3 show the results.

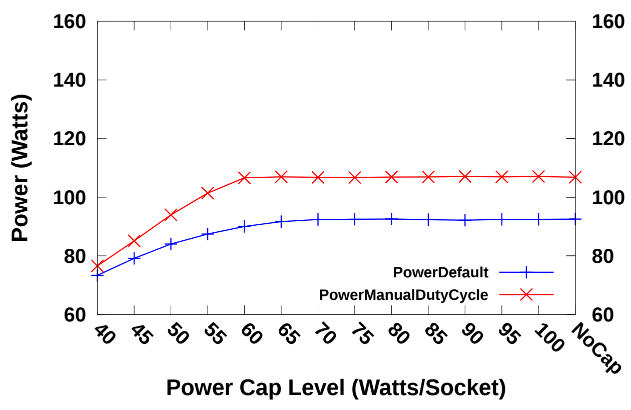


Fig. 2. Power

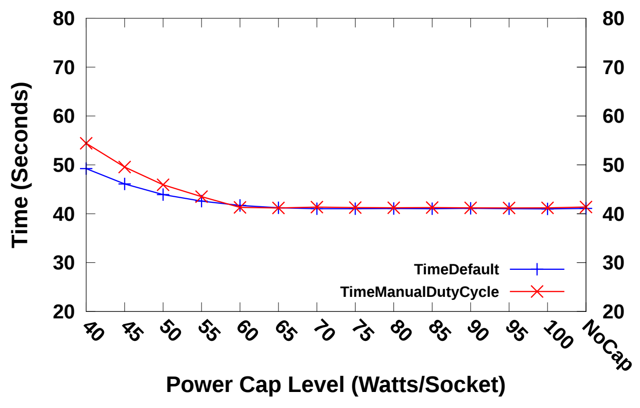


Fig. 3. Time

It shows that OpenMP program did not get benefits from slowing down some of the treads .

When using icc, in the case of No Cap, the results are similar: manual duty cycle: Application(EnergyStat) - Time 22.847241 Total energy consumed 2298.958635 Ave. Power Level 100.623032 Final Temperature socket 1 : 46 socket 2 : 37 original (wo duty cycle) Application(EnergyStat) - Time 23.059587 Total energy consumed 2125.018620 Ave. Power Level 92.153369 Final Temperature socket 1 : 46 socket 2 : 39 Again, here the power level increase!

When I comment out the `do_wait` function in GOMP (which is equivalent to increase GOMP_SPIN_COUNT– default 3ms, very short) The graph looks like the following:

In light of the difference, trying to set duty cycle in a destroyed world, actually could save some power. But the power is still higher than the default (see the following graphs). It seems that for OpenMP this approach has no hope. Thus, community detection code cannot benefit from such technique, although MPI seems to have benefited a lot.

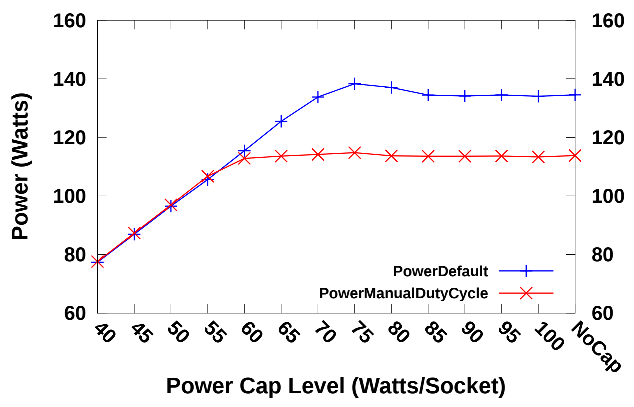


Fig. 4. Power

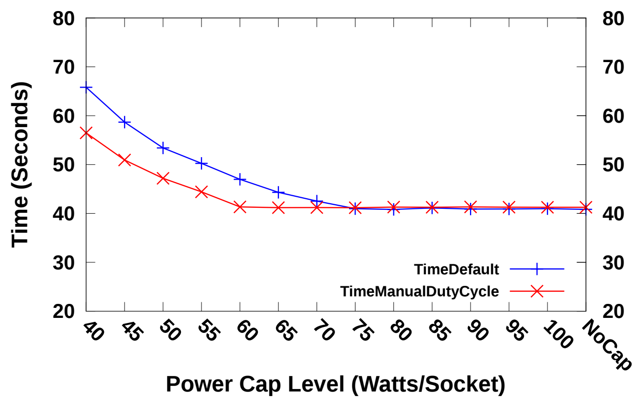


Fig. 5. Time

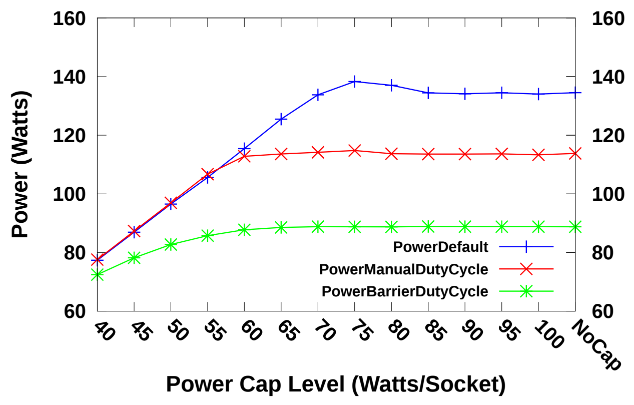
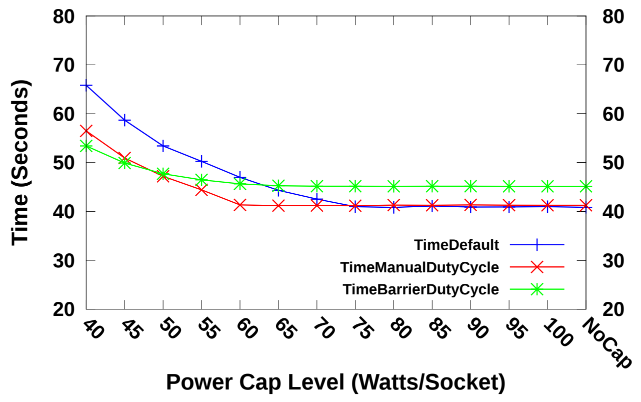


Fig. 6. Power



Time (Seconds)
Power Cap Level (Watts/Socket)
TimeDefault
TimeManualDutyCycle
TimeBarrierDutyCycle
20
30
40
50
60
70
80
40
45
50
55
60
65
70
75
80
85
90
95
100
NoCap

Fig. 7. Time