A2E: Adaptively Aggressive Energy Efficient DVFS Scheduling for Data Intensive Applications

Li Tan¹, Zizhong Chen¹, Ziliang Zong², Rong Ge³, and Dong Li⁴
¹University of California, Riverside
²Texas State University-San Marcos
³Marquette University
⁴Oak Ridge National Laboratory

IPCCC'13, San Diego, California, USA
Dec. 6, 2013
Power Management via DVFS

- Power and energy consumption of high performance computing is a growing severity → operating costs and system reliability.

- Dynamic Voltage and Frequency Scaling (DVFS)
  - voltage/frequency ↓ → power ↓ → energy efficiency
  - Peak CPU performance is not necessary when slack exists: load imbalance, network latency, communication delay, memory and disk access stalls, etc.
Power Management via DVFS (Cont.)

- Types of Workloads (A real app. is often *hybrid*.)
  - Computation (*compute intensive*)
  - Communication (*communication intensive*)
  - Memory accesses and disk accesses (*data intensive*)

- Energy Efficient DVFS Scheduling Strategies
  - Computation: Peak CPU perf. is always needed
  - Communication: Volt./Freq. ↓ during communication
    - Peak Volt./Freq. during computation
  - Data Accesses: Non-intuitive/difficult and costly
    - Hard to separate out workloads + high DVFS overhead
Code Example

1: while (caseA) {
2:   ...
3:   buffer = (char*)malloc(num*sizeof(char));
4:   /* MPI communication routine call I */
5:   MPI_Bcast(&buffer, count, type, root, comm);
6:   /* Independent computation code */
7:   computation();
8:   /* MPI communication routine call II */
9:   MPI_Alltoall(&sb, sc, st, &rb, rc, rt, comm);
10:  ...
11: }

Fig. 3. Typical Kernel Pattern of Communication Intensive Code.

1: while (caseA) {
2:   ...
3:   /* Memory accesses mixed with computation */
4:   valueA = arrayA[baseA+offset];
5:   arrayB[baseB] += valueB;
7:   ...
8:   /* Disk accesses mixed with computation */
9:   buffer = (char*)malloc(num*sizeof(char));
10:  fread(buffer, size, count, read_file_stream);
11:  fwrite(buffer, size, count, write_file_stream);
12:  ...
13: }

Fig. 4. Typical Kernel Pattern of Memory and Disk Access Intensive Code.
Energy Saving Block (ESB)

- Motivated by the term *basic block* in the area of *compilers*.

- Definition
  - A *statement block* of one specific type of workload
  - Comp-ESB, Comm-ESB, Mem-ESB, and Disk-ESB
  - Runtime energy savings may be achieved by DVFS

- Energy saving opportunities can be exploited between the *boundary* of different ESBs
Code Example (Cont.)

```c
1: while (caseA) {
2:   ...
3:   buffer = (char*)malloc(num*sizeof(char));
4:   /* MPI communication routine call I */
5:   MPI_Bcast(&buffer, count, type, root, comm);
6:   /* Independent computation code */
7:   computation();
8:   /* MPI communication routine call II */
9:   MPI_Alltoall(&sb, sc, st, &rb, rc, rt, comm);
10:  ...
11: }
```

Fig. 3. Typical Kernel Pattern of Communication Intensive Code.

```c
1: while (caseA) {
2:   ...
3:   /* Memory accesses mixed with computation */
4:   valueA = arrayA[baseA+offset];
5:   arrayB[baseB] += valueB;
7:   ...
8:   /* Disk accesses mixed with computation */
9:   buffer = (char*)malloc(num*sizeof(char));
10:  fread(buffer, size, count, read_file_stream);
11:  fwrite(buffer, size, count, write_file_stream);
12:  ...
13: }
```

Fig. 4. Typical Kernel Pattern of Memory and Disk Access Intensive Code.
Basic DVFS Scheduling Strategy

1: while (case) {
2:     …
3:         SetFreq(Low);
4:     communication();
5:         SetFreq(High);
6:     memory_access();
7:     disk_access();
8:     computation();
9:         SetFreq(Low);
10:    communication();
11:    SetFreq(High);
12:    …
13: }

**Basic Idea:**
- Communication is not CPU-bound
- Schedule the lowest V/F for comm.
- Schedule the highest V/F for comp.

**Disadvantages**
- Only works at inter-ESB level while fails at intra-ESB level, i.e., cannot save energy for Mem-/Disk-ESBs
- Number of CPU frequency switches can be considerably large → high DVFS overhead (time and energy)
Aggressive DVFS Scheduling Strategy (AGGREE)

```
SetFreq(Low);
1: while (case) {
2:   ... 
3:   communication();
4:   memory_access();
5:   disk_access();
6:   computation();
7:   communication();
8:   ... 
9: }
SetFreq(High);
```

**Basic Idea:**
- For a loop of ESBs with a small proportion of computation (Comm-ESB, Mem-ESB, and Disk-ESB)
  - Aggressively V/F↓ for the whole loop
    - minor perf. ↓ + major energy ↓
  - The number of frequency switches ↓
- **Disadvantages**
  - Performance loss trade-off can be further moderated for higher energy-performance efficiency
Adaptively Aggressive DVFS Scheduling Strategy (A2E)

```c
SetFreq(Low);
1: while (case) {
2:     ... 
3:     communication();
4:     SetFreq(Medium1);
5:     memory_access();
6:     SetFreq(Medium2);
7:     disk_access();
8:     SetFreq(High);
9:     computation();
10:    communication();
11:    ... 
12: } 
13: SetFreq(High);
```

**Basic Idea:**
- Moderate low-performance trade-off
- Set an intermediate V/F adaptively
- Based on the proportion of comp. time among the total execution time
- Aggressively set the V/F once for Mem-ESB and Disk-ESB as a whole

**Advantages over the previous two**
- Integrate the strengths of both
- Achieve the optimal energy-performance efficiency
Adaptively Aggressive DVFS Scheduling Algorithm

**Example:** (See the paper for algorithm details.)
Consider a data intensive application with 10 ESBs

- The highest *proportion* of *computation time* is 20%
  - We empirically obtain in advance the comp. time %
- There exist 4 *gears* of CPU frequency for DVFS
  - $f_0$, $f_1$, $f_2$, and $f_3$ (assume $f_0 < f_1 < f_2 < f_3$)
- Consequently, 4 *sub-ranges* for adaptively aggressive DVFS scheduling by *comp. proportion*
  - [0, 5%) $\rightarrow f_0$
  - [5%, 10%) $\rightarrow f_1$; [10%, 15%) $\rightarrow f_2$; [15%, 20%] $\rightarrow f_3$.
Speculative DVFS Scheduling Strategy

**Basic Idea:**
- Speculation is a *compiler* technique for predicting instruction’s execution
- Speculate the outcome of imbalanced branches
- Set the lowest f for the whole loop
- Set the highest f for computation inside the rarely taken branch as recovery for *mis-speculation*

**Advantages**
- The number of frequency switches ↓
- Performance is not traded off

```c
SetFreq(Low);
1: while (caseA) {
2:   if (caseB) { P1
3:     ...  
4:       SetFreq(Low);
5:       communication();
6:     }  
7:   else { P2 (P2 <= P1)
8:     ...  
9:       SetFreq(High);
10:   }  
11: }  
12: SetFreq(High);
```
Implementation

- Energy Efficient DVFS Scheduling Strategies
  - Basic → can only handle Comm-ESB
  - AGGREE → can handle Comm-ESB, Mem-ESB, and Disk-ESB
  - A2E → can handle Comm-ESB, Mem-ESB, and Disk-ESB, with moderated performance trade-off
  - Speculation → applied to both AGGREE and A2E

- Applicable Applications
  - Data (memory and disk access) intensive applications with imbalanced branches
Evaluation

- **Benchmarks**
  - Applied all strategies to 5 high performance data intensive benchmarks
  - Selected from NPB and ASC benchmark suites

- **Hardware Configuration**
  - A power-aware cluster comprised of 8 computing nodes with two Quad-core 2.5 GHz AMD Opteron 2380 processors (*Freq*: {0.8, 1.3, 1.8, 2.5} GHz)
    - Totalling 64 cores, energy measured by PowerPack
  - 8 GB RAM per node, 64-bit Linux kernel 2.6.32
Performance Loss and Energy Savings

DT (Class B) from NPB Benchmark Suite

MG (Class C) from NPB Benchmark Suite

CPU Frequency Scheduling Strategy

SPhot (Track 4000 Particles) from ASC Benchmark Suite

MP1BZIP2 (Compress a 0.77GB file) from bzip2 Benchmark

CPU Frequency Scheduling Strategy

cp_MPI (Copy a 54.4MB file) from Linux Kernel

CPU Frequency Scheduling Strategy
Energy Savings for Imbalanced Branches
Energy and Performance Efficiency Trade-off

**EDP**: Energy Delay Product; **ED2P**: Energy Delay-Squared Product

- Two useful metrics to evaluate the balance between energy and performance efficiency
Conclusions

Adaptively Aggressive Energy Saving for Data Intensive Applications (Mem. and Disk Access)

Novelty

- Overcome the disadvantages of other approaches to save energy for app. w/ mixed types of workloads
- Achieve the optimal energy and performance efficiency by moderating performance loss trade-off
- Save extra energy for imbalanced branches by spec.
- Experimentally on average 32.6% energy savings with 6.2% performance loss for 5 real applications