A2E: *A*daptively *A*ggressive *E*nergy Efficient DVFS Scheduling for Data Intensive Applications

Li Tan[1], Zizhong Chen[1], Ziliang Zong[2], Rong Ge[3], and Dong Li[4]

[1] *University of California, Riverside*

[2] *Texas State University-San Marcos*

[3] *Marquette University*

[4] *Oak Ridge National Laboratory*

UCR UNIVERSITY OF CALIFORNIA, RIVERSIDE

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

```
while (caseA) {
  ...
  buffer = (char*)malloc(num*sizeof(char));
  /* MPI communication routine call I */
  MPI_Bcast(&buffer, count, type, root, comm);
  /* Independent computation code */
  computation();
  /* MPI communication routine call II */
  MPI_Alltoall(&sb, sc, st, &rb, rc, rt, comm);
  ...
}
```

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

```
while (caseA) {
  ...
  /* Memory accesses mixed with computation */
  valueA = arrayA[baseA+offset];
  arrayB[baseB] += valueB;
  arrayC[baseC++] = arrayB[baseB++]+valueC;
  ...
  /* Disk accesses mixed with computation */
  buffer = (char*)malloc(num*sizeof(char));
  fread(buffer, size, count, read_file_stream);
  fwrite(buffer, size, count, write_file_stream);
  ...
}
```

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.)

- - - - - - ESB boundary

```
while (caseA) {
  ...
  buffer = (char*)malloc(num*sizeof(char));
  /* MPI communication routine call I */
  MPI_Bcast(&buffer, count, type, root, comm);
  /* Independent computation code */
  computation();
  /* MPI communication routine call II */
  MPI_Alltoall(&sb, sc, st, &rb, rc, rt, comm);
  ...
}
```

Explicit Boundary

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

```
while (caseA) {
  ...
  /* Memory accesses mixed with computation */
  valueA = arrayA[baseA+offset];
  arrayB[baseB] += valueB;
  arrayC[baseC++] = arrayB[baseB++]+valueC;
  ...
  /* Disk accesses mixed with computation */
  buffer = (char*)malloc(num*sizeof(char));
  fread(buffer, size, count, read_file_stream);
  fwrite(buffer, size, count, write_file_stream);
  ...
}
```

Implicit Boundary

Fig. 4. Typical Kernel Pattern of Memory and Disk Access Intensive Code.

Basic DVFS Scheduling Strategy

```
while (case) {
    …
    SetFreq(Low);
    communication();
    SetFreq(High);
    memory_access();
    disk_access();
    computation();
    SetFreq(Low);
    communication();
    SetFreq(High);
    …
}
```

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);
while (case) {
    …
    communication();
    memory_access();
    disk_access();
    computation();
    communication();
    …
}
   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)

```
SetFreq(Low);
while (case) {
    …
    communication();
    SetFreq(Medium1);
    memory_access();
    SetFreq(Medium2);
    disk_access();
    SetFreq(High);
    computation();
    SetFreq(Low);
    communication();
    …
}
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%) → f_0;
  - [5%, 10%) → f_1; [10%, 15%) → f_2; [15%, 20%] → f_3.

Speculative DVFS Scheduling Strategy

```
SetFreq(Low);
while (caseA) {
    if (caseB) { P1
        …
        SetFreq(Low);
        communication();
        SetFreq(High);
        …
    }
    else { P2 (P2 << P1)
        …
        SetFreq(High);
        computation();
        …
    }
}
SetFreq(High);
```

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

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*

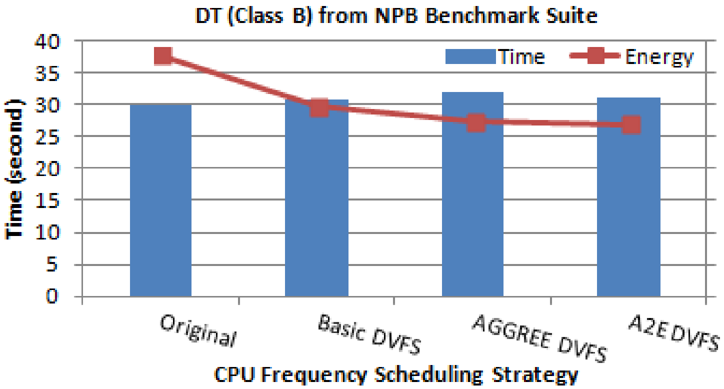
- Applicable Applications
  - Data (memory and disk access) intensive applications with imbalanced branches

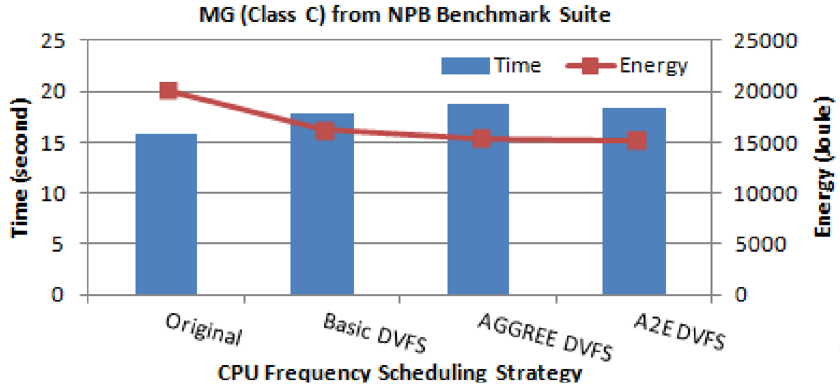
Evaluation

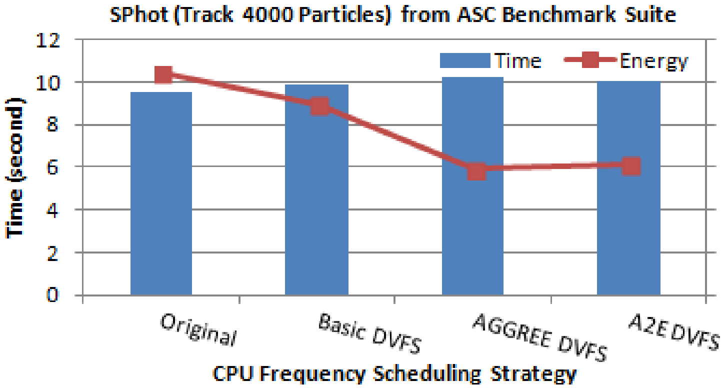
- Benchmarks
  - Applied all strategies to 5 high performance data intensive benchmarks
  - Selected from NPB and ASC benchmark suites

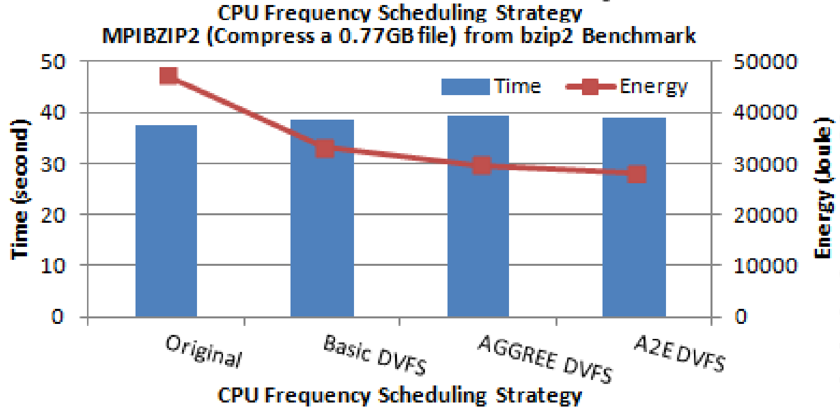
- Hardware Configuration
  - A power-aware cluster comprised of 8 computing nodes with two Quad-core 2.5 GHz AMD Opteron 2380 processors (*Freq*: {0.8, 1.3, 1.8, 2.5} GHz)
    - Totalling 64 cores, energy measured by PowerPack
  - 8 GB RAM per node, 64-bit Linux kernel 2.6.32

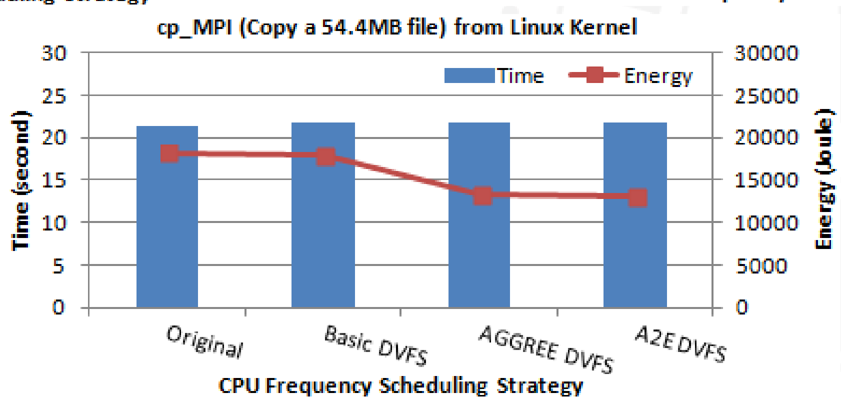
Performance Loss and Energy Savings



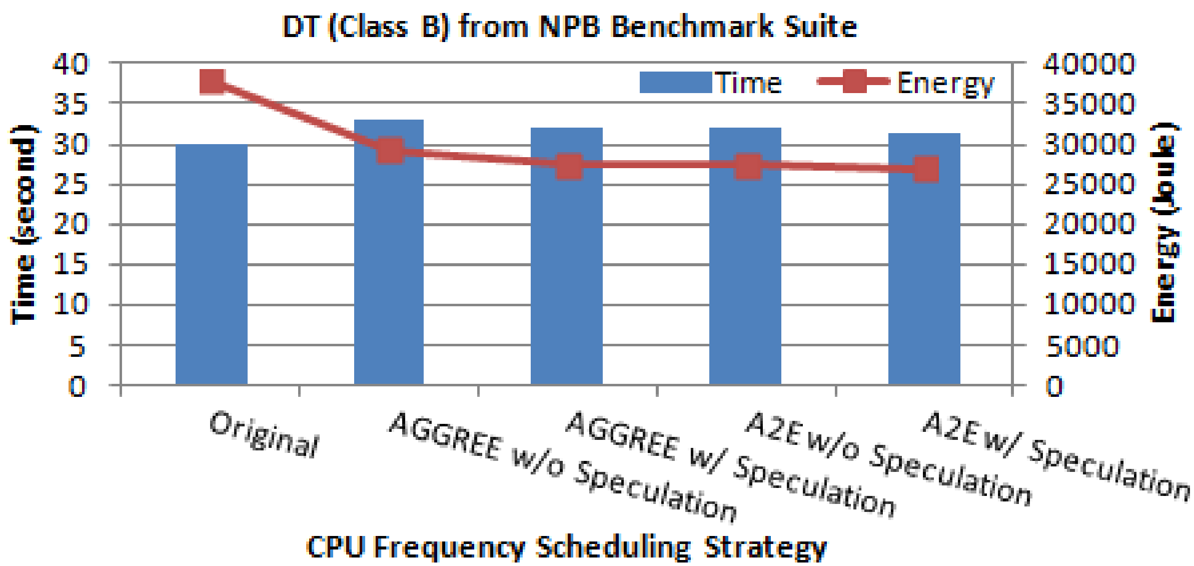




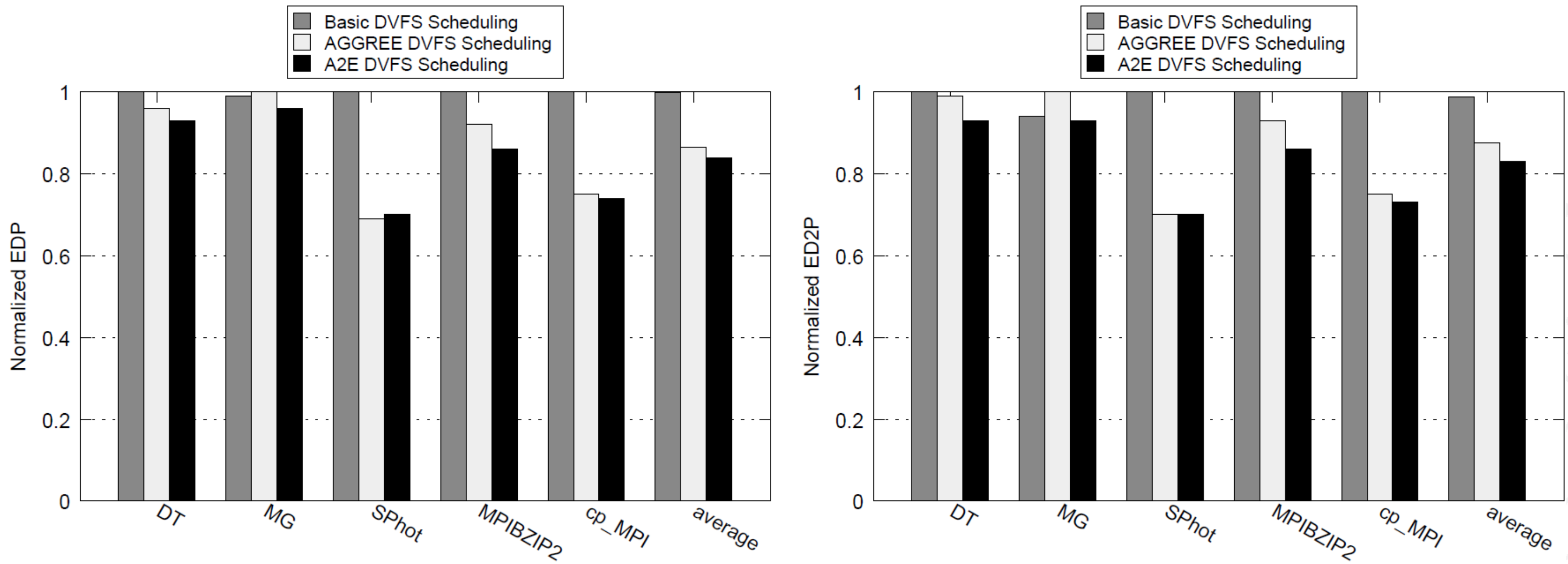




Energy Savings for Imbalanced Branches



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