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

Li Tan and Zizhong Chen
University of California, Riverside
{ltan003, chen}@cs.ucr.edu

Ziliang Zong
Texas State University-San Marcos
zz11@txstate.edu

Rong Ge
Marquette University
rong.ge@marquette.edu

Dong Li
Oak Ridge National Laboratory
lid1@ornl.gov

***Abstract*—Featured by high portability and programmability, Dynamic Voltage and Frequency Scaling (DVFS) has been widely employed to achieve energy efficiency for high performance applications on distributed-memory architectures nowadays through various scheduling algorithms. Generally, different forms of slack from load imbalance, network latency, communication delay, memory and disk access stalls, etc. are exploited as energy saving opportunities where peak CPU performance is not necessary, with little or limited performance loss. The deployment of DVFS for communication intensive applications is straightforward due to the explicit boundary between Energy Saving Blocks (ESBs) at source code level, while for data (e.g., memory and disk access) intensive applications it is difficult for applying DVFS since ESB boundary is implicit due to mixed types of workloads. We propose an adaptively aggressive DVFS scheduling strategy to achieve energy efficiency for data intensive applications, and further save energy via speculation to mitigate DVFS overhead for imbalanced branches. We implemented and evaluated our approach using five memory and disk access intensive benchmarks with imbalanced branches against another two energy saving approaches. The experimental results indicate an average of 32.6% energy savings were achieved with 6.2% average performance loss compared to the original executions on a power-aware 64-core cluster.**

***Keywords*—energy; performance; DVFS; adaptive; aggressive; speculative; data intensive, memory accesses, disk accesses.**

## I. Introduction

With the growing severity of power and energy consumption on high performance distributed-memory computing systems nowadays in terms of operating costs and system reliability [1] [2], reducing power and energy costs has been considered as a critical issue in high performance computing, in particular in this big data era [3]. Featured by high portability and programmability, Dynamic Voltage and Frequency Scaling (DVFS) [4] [5] techniques have been empirically applied for scaling down power and energy costs with little or limited performance loss [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16]. Generally, energy efficiency can be achieved during runs of high performance applications by scaling down operating voltage and frequency of CPU, where peak CPU performance is not necessary such as slack from load imbalance, communication delay, memory and disk access latency, etc., given the assumption that CPU dominates the total system-wise energy consumption. DVFS is thus deemed an effective approach to address the concerns of operating costs and system reliability for high performance applications nowadays.

Per the functionality of an application, types of workloads within the application consist of computation, communication, memory accesses, and disk accesses, etc. For communication intensive applications, an effective way of improving energy efficiency, referred to as basic DVFS scheduling strategy, is to scale down CPU voltage and frequency during communication, while keep peak CPU performance when CPU is fully loaded during computation. This approach can be easily fulfilled, since at source code level the boundary between communication and computation is explicit. Appropriate CPU frequency can be assigned via DVFS techniques at the boundary between communication and computation. Since the execution of communication is not CPU-bound, communication time will barely increase due to low CPU performance. Moreover, computation time will not grow since CPU performance during computation is kept the same as the orignal by not altering CPU frequency. Since generally voltage is proportional to frequency, energy savings can be achieved using basic DVFS scheduling strategy with negligible performance loss due to lower CPU voltage and frequency on average compared to the original execution.

Similarly, peak CPU performance is not needed when CPU is waiting for data from memory and disk. Typically, for memory and disk access intensive applications, memory and disk access latency are performance bottleneck of the applications. According to the fundamental memory hierachy of moderm computer architectures, compared to CPU, main memory access takes hundreds of clock cycles while local disk access time is of the order of magnitude of millisecond, $10^6$ greater than memory access time in general. As for memory and disk access intensive applications, energy efficiency can be intuitively achieved by reducing CPU frequency when memory and disk accesses are performed and CPU is waiting for data.

Despite the straightforward deployment of DVFS for communication intensive applications, it is however not intuitive to achieve energy efficiency for other types of data intensive applications such as memory and disk access intensive applications due to two reasons: Firstly, employing DVFS in our approach is implemented at source code level within the application via system calls for modifying CPU frequency configuration files at runtime. Empirically, memory and disk accesses are generally accompanied by CPU-bound operations at source code level, which causes the boundary between memory and disk accesses and computation implicit. As a consequence, it is difficult to separate memory and disk

**978-1-4799-3214-6/13/$31.00 ©2013 IEEE**

accesses from computation and then apply DVFS for energy savings. Secondly, the overhead on employing DVFS can be high: Given the iterative nature of many high performance applications, the time and energy costs on employing fine-grained DVFS scheduling can be non-negligible due to a large number of CPU frequency switches [12] [17] [18]. A lightweight DVFS scheduling strategy is thus desirable.

In this paper, we introduce an adaptively aggressive DVFS scheduling strategy (A2E) for energy efficient memory and disk access intensive applications with imbalanced branches, where memory and disk accesses are mixed with minor computation. Instead of separating memory and disk accesses from computation for an Energy Saving Block (ESB) with different types of workloads, and then performing fine-grained DVFS scheduling accordingly, we aggressively apply DVFS to the hybrid ESB holistically, and adaptively set an appropriate CPU frequency to the hybrid ESB according to the computation time proportion within the total execution time of the ESB. In summary, the contributions of this paper are as follows:

- We analyze the impact of factors such as CPU frequency and execution time on energy consumption of applications consisting of different dominant workloads, which motivates our idea of A2E;
- We demonstrate the significance of code boundary for achieving energy efficiency via DVFS, and thus define ESB to refine energy saving opportunities and model energy and performance efficiency of our approach;
- We propose A2E to improve energy efficiency for memory and disk access intensive applications with mixed minor computation, and further save energy using speculation to mitigate DVFS overhead for imbalanced branches. Our approach is evaluated to achieve considerable energy savings (32.6% on average) and incur minor performance loss (6.2% on average) compared to the original runs of five benchmarks.

The rest of this paper is organized as follows. Section 2 discusses relevant research. Section 3 motivates and section 4 introduces three energy saving approaches for data intensive applications. We provide implementation details and evaluate our approach in section 5, and section 6 concludes.

## II. Related Work

**DVFS Scheduling for Compute Intensive Applications**: A large body of work has been done for achieving energy efficiency in compute intensive applications by exploiting CPU slack or idle time from imbalanced CPU-bound applications. Ge *et al.* [13] proposed a runtime system and an integrated performance model for achieving energy efficiency and constraining performance loss through DVFS and performance modeling and prediction. Rountree *et al.* [15] presented another runtime system by improving and extending previous classic scheduling algorithms and achieved significant energy savings with extremely limited performance loss. Kappiah *et al.* [8] proposed a scheduled iteration method that computes the total slack per processor per timestep, then scheduling CPU frequency for the upcoming timestep.

**DVFS Scheduling for Data Intensive Applications**: There also exists a large amount of work for energy efficient communication via different DVFS scheduling algorithms. A relatively small amount of research has been conducted for reducing energy costs of memory/disk access intensive applications. Kappiah *et al.* [8] devised a system that exploits slack arising at synchronization points of MPI programs by reducing inter-node energy gear via DVFS. Li *et al.* [19] proposed to characterize energy saving opportunities in executions of hybrid MPI/OpenMP applications without performance loss. Predictive models and novel algorithms were presented via statistical analysis of power and time requirements under different configurations. Ge *et al.* [1] observed that memory stalls in the memory-bound sequential application swim from the SPEC CPU2000 benchmark suite produced considerable slack for energy savings via DVFS with alomost no impact on performance. Our work focuses on improving energy efficiency for parallel exeuctions of memory/disk-bound applications on distributed-memory computing systems.

**Aggressive and Speculative Mapping and Scheduling**: Liu *et al.* [20] leveraged the fact that at runtime some applications typically have shorter execution time than their worst-case execution time, and applied DVFS to dynamically and aggressively reduce voltage and frequency on a heterogeneous system consisting of CPUs and GPUs. Luo *et al.* [21] proposed to improve energy efficiency for thread-level speculation in a same-ISA heterogeneous multicore system with an overhead throttling mechanism and a competent resource allocation scheme. Our work differs from them in that prior knowledge of the worst-case execution time of the application is not a prerequisite, and the target of our work is data intensive applications running on a distributed-memory architecture.

## III. Motivation: DVFS Scheduling for Different Workload Intensive Applications

In order to learn the impact of factors such as CPU frequency and execution time that may affect energy consumption of applications with different dominant workloads, we conducted some experiments and the results are plotted in Figures 1 and 2. Motivated by the experimental results on DVFS scheduling for compute intensive and compute/non-compute comparable applications, we observe that the proportion of non-compute operations in an application determines whether energy consumption of the application is time-directed or frequency-directed. In other words, energy consumption is affected more by exectuion time in a compute intensive application, and is affected more by CPU frequency in a compute/non-compute comparable application, given the fact that energy consumption equals product of average power and time, where power is proportional to frequency and voltage.

As shown in Figure 1, CPU performance degradation for the compute intensive application EP from NPB [22] leads to more energy costs due to the longer execution time that is the more dominant factor compared to CPU frequency. On the other hand, for an application with comparable proportion of computation and non-computation such as pdgemm() routine from ScaLAPACK [18] shown in Figure 2, there exists even a slight decrease of energy costs as CPU frequency goes down, despite the increasing execution time due to low CPU performance. We can further infer from the experimental results that if computation only takes a small proportion of the total execution time of an application as in the case of data

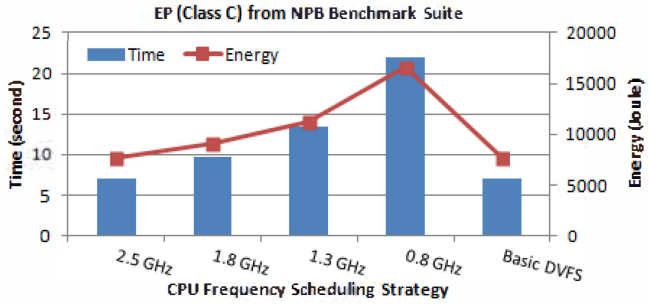


Fig. 1. DVFS Scheduling for Compute Intensive Application.

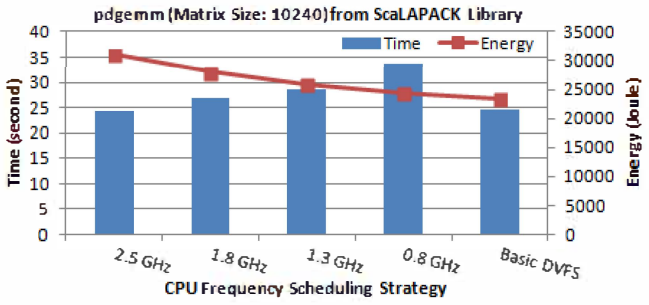


Fig. 2. DVFS Scheduling for Compute/Non-Compute Comparable Application.

```
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;
6:   arrayC[baseC++] = arrayB[baseB++]+valueC;
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.

intensive applications such as memory and disk access intensive applications, performance loss at a low CPU frequency is comparatively limited, and the less computation exists, the less performance loss is incurred from reducing CPU frequency. Therefore, the resulting reduction of power from lowering frequency dominates the ultimate energy costs. Compared to the compute/non-compute comparable application, more energy savings can be achieved by aggressively reducing CPU frequency for a non-compute intensive application.

Non-compute intensive applications can be any applications with a dominant proportion of non-compute workloads such as communication, memory accesses, and disk accesses, etc., where memory and disk access intensive applications are commonly regarded as data intensive applications. Different from communication intensive applications, it is challenging to employ DVFS on memory and disk access intensive applications for achieving energy efficiency, since data operations such as memory and disk accesses generally mix with minor computation at source code level. As we know, energy savings can be achieved by applying DVFS at source code level by lowering CPU frequency for data intensive operations where peak CPU performance is not necessary. It is however difficult to separate non-computation from computation for later assignment of appropriate CPU frequency to different workloads. To fulfill energy efficiency for data intensive applications, our goals include: (a) Reducing the performance loss from computation accompanying data intensive operations due to low CPU frequency, i.e., low-performance trade-off; (b) reducing the number of CPU frequency switches by DVFS, i.e., DVFS overhead. Both low-performance trade-off and DVFS overhead result in higher execution time and thus greater energy costs.

## IV. Energy Efficient DVFS Scheduling Strategies for Data Intensive Applications

In this section, we present our adaptively aggressive energy efficient DVFS scheduling strategy (A2E) for data intensive applications, e.g., memory and disk access intensive applications. Leveraging speculation, A2E can also handle conditional statements with imbalanced branches whose possibilities of occurrence are significantly different. Next we first introduce the concept of Energy Saving Blocks at source code level.

### A. Energy Saving Blocks

Similarly as the common term *basic block* in the area of compilers, from the perspective of energy, an Energy Saving Block (ESB) is defined as a statement block of one specific type of workload such as computation, communication, memory accesses and disk accesses, etc., where runtime enery savings may be achieved by different means. For simplicity, such ESBs are referred to as *Comp*-ESB, *Comm*-ESB, *Mem*-ESB, and *Disk*-ESB respectively in the later text. For instance, in the code example shown in Figure 5 (a), there exist six ESBs located at Lines 5, 7, 8, 9, 11, and 17, respectively, i.e., two *Comp*-ESBs, two *Comm*-ESBs, one *Mem*-ESB, and one *Disk*-ESB, each of which can be assigned an appropriate CPU frequency accordingly via DVFS for energy saving purposes.

### B. Basic DVFS Scheduling for Comp-ESB and Comm-ESB

We can apply a basic DVFS scheduling strategy for *Comp*-ESB and *Comm*-ESB that simply sets CPU frequency to as high as possible for *Comp*-ESB and sets CPU frequency to as low as possible for *Comm*-ESB, which can be easily fulfilled since the boundary of *Comm*-ESB is explicit as shown in Figure 3: Little computation is involved in the MPI communication routine calls at Lines 5 and 9 respectively, and computation independent of communication at Line 7 is conducted after the communication code. The basic DVFS scheduling strategy is shown in Figure 5 (a), where a low-high CPU frequency pair is assigned around the communication code, since CPU is barely utilized in the communication and peak CPU performance is thus not necessary. Yet, the basic

```
1: while (caseA) {
2:   if (caseB) { P1
3:     ...
4:     SetFreq(L_DVFS);
5:     communication();
6:     SetFreq(H_DVFS);
7:     memory_access();
8:     disk_access();
9:     computation();
10:    SetFreq(L_DVFS);
11:    communication();
12:    SetFreq(H_DVFS);
13:    ...
14:  }
15:  else { P2 (P2 ≪ P1)
16:    ...
17:    computation();
18:    ...
19:  }
20: }
```

(a) Basic DVFS Scheduling

```
1: SetFreq(L_DVFS);
2: while (caseA) {
3:   if (caseB) { P1
4:     ...
5:     communication();
6:     memory_access();
7:     disk_access();
8:     computation();
9:     communication();
10:    ...
11:  }
12:  else { P2 (P2 ≪ P1)
13:    ...
14:    SetFreq(H_DVFS);
15:    computation();
16:    SetFreq(L_DVFS);
17:    ...
18:  }
19: }
20: SetFreq(H_DVFS);
```

(b) Aggressive DVFS Scheduling with Speculation (AGGREE)

Fig. 5. Basic and Aggressive DVFS Scheduling for Typical Communication, Memory Access, and Disk Access Mixed Code with Imbalanced Branches.

DVFS scheduling strategy suffers from two disadvantages: (a) It can only work at inter-ESB level but fail at intra-ESB level, i.e., towards single ESB with mixed workloads as shown in Figure 4 (we discuss it next); (b) the number of CPU frequency switches can be considerably large if the number of *Comm*-ESBs and the number of iterations of the loop are large, which incurs non-negligible overhead on time and energy [18].

### C. Aggressive DVFS Scheduling for Mem-ESB and Disk-ESB

Figure 4 depicts typical kernel of memory and disk access intensive applications. Lines 4, 5, and 6 give three typical memory accesses mixed with computation. At Line 4, `valueA` is assigned until the finish of calculating the array index and accessing the content of corresponding memory location. Lines 5 and 6 show how array values are involved in computation after and before addressing, respectively. Likewise, for disk accesses given at Lines 10 and 11 that read and write blocks of data from and into local disk files individually, the value of input/output buffer pointer is frequently accessed and updated for current and next reading/writing position as the file reading and writing operations proceed. If the CPU-bound computation time is significant among the total execution time of the *Mem*-ESB/*Disk*-ESB, i.e., in the case of compute intensive applications, considerable slowdown will be incurred from reducing CPU frequency for the ESB as a whole, and thus energy consumption grows as the trend shown in Figure 1.

Yet for applications with a small proportion of computation mixed with memory and disk accesses depicted in Figure 4, aggressively reducing CPU frequency for the whole ESB only causes minor performance loss while obtains considerable energy savings from low CPU frequency and voltage during waiting for memory and disk data, since memory and disk access time dominate the total execution time. Basic DVFS scheduling strategy fails to achieve energy savings for such applications since it is difficult to separate non-computation from computation and then apply DVFS accordingly. Even if the programmer manages to rewrite the source code for categorizing ESBs with explicit boundary between each other via the use of temporary variables, etc. (we use this method to calculate the proportion/percentage of different types of workloads within a hybrid ESB), performance and energy loss can be caused by numerous CPU frequency switches within the loop of ESBs, as shown in Figure 5 (a), the kernel of an application with different types of workloads including computation, communication, memory accesses and disk accesses. The basic DVFS scheduling strategy sets CPU frequency to low before the *Comm*-ESBs at Lines 5 and 11 respectively and sets it back to high after the *Comm*-ESBs. It keeps CPU frequency high for all *Mem*-ESB, *Disk*-ESB, and *Comp*-ESB if the *Mem*-ESB and the *Disk*-ESB are accompanied by minor computation as shown in Figure 4. Potential energy saving opportunities can be leveraged by aggressive DVFS scheduling (AGGREE) as presented in Figure 5 (b). Instead of fine-grained deployment of DVFS for setting appropriate CPU frequency to *Comm*-ESBs without exploiting energy saving opportunities from *Mem*-ESBs and *Disk*-ESBs, AGGREE aggressively sets CPU frequency to low once for the whole loop given that the loop is data intensive, which achieves higher energy efficiency than the basic DVFS scheduling strategy due to lower CPU power at the cost of minor performance and energy loss from the small proportion of computation. Moreover, AGGREE overcomes the excessive number of CPU frequency switches by croase-grained DVFS scheduling outside the loop.

**Algorithm 1** *Adaptively Aggressive DVFS Scheduling Algo.*

SetDVFS($ESB$, $p_{comp}$) /*Assume $f_0 < f_1 < \cdots < f_{N_f-1}$*/

```
1:  Bcast(p_comp)
2:  N_f ← GetNumFreq()
3:  p'_comp ← Max(p_comp of all ESBs)
4:  pSet_{0,...,N_f-1} ← GetRange(p'_comp, N_f)
5:  while 0 ≤ i < N_f − 1 do
6:    if (0 ≤ p_comp < pSet_i) then
7:      SetFreq(f_0)
8:    else if (pSet_i ≤ p_comp < pSet_{i+1}) then
9:      SetFreq(f_i)
10:   else if (p_comp ≥ pSet_{N_f-1}) then
11:     SetFreq(f_{N_f-1})
12:   end if
13:   i ← i + 1
14: end while
```

### D. Adaptively Aggressive DVFS Scheduling for Mem-ESB and Disk-ESB

Recall one of our goal is to reduce the performance loss from minor computation accompanying data intensive operations at low CPU frequency. One effective way to moderate the low-performance trade-off from AGGREE for data intensive applications is to set an intermediate CPU frequency adaptively on case-by-case basis for *Mem*-ESBs and *Disk*-ESBs within such applications, instead of always employing the lowest CPU frequency during executions. We refer to this adaptively aggressive DVFS scheduling strategy as A2E. The heuristic of A2E is similar to AGGREE: For those ESBs with implicit boundaries, we specify an appropriate CPU frequency for them as a whole, since fine-grained DVFS scheduling upon the finish of separating non-computation from computation is difficult. Considering the code example shown in Figure 5 (b), AGGREE aggressively sets CPU frequency to the lowest possible value once outside the data intensive loop, while A2E calculates an intermediate CPU frequency adaptively

```
1: SetFreq(L_DVFS);
2: while (caseA) {
3:   if (caseB) { P1
4:     ...
5:     communication();
6:     memory_access();
7:     disk_access();
8:     computation();
9:     communication();
10:    ...
11:  }
12:  else { P2 (P2 ≪ P1)
13:    ...
14:    SetFreq(H_DVFS);
15:    computation();
16:    SetFreq(L_DVFS);
17:    ...
18:  }
19: }
20: SetFreq(H_DVFS);
```

(a) Aggressive DVFS Scheduling with Speculation (AGGREE)

```
1: while (caseA) {
2:   if (caseB) { P1
3:     ...
4:     SetFreq(L_DVFS);
5:     communication();
6:     SetFreq(M_DVFS);
7:     memory_access();
8:     SetFreq(M'_DVFS);
9:     disk_access();
10:    SetFreq(H_DVFS);
11:    computation();
12:    SetFreq(L_DVFS);
13:    communication();
14:    ...
15:  }
16:  else { P2 (P2 ≪ P1)
17:    ...
18:    SetFreq(H_DVFS);
19:    computation();
20:    ...
21:  }
22: }
```

(b) Adaptively Aggressive DVFS Scheduling with Speculation (A2E)

Fig. 6. AGGREE and A2E DVFS Scheduling for Typical Communication, Memory Access, and Disk Access Mixed Code with Imbalanced Branches.

according to the proportion of computation time among the total execution time of an ESB, and also aggressively sets the calculated frequency once for the ESB with mixed workloads. Algorithm 1 details the steps of employing A2E. For each ESB in the application, we empirically obtain in advance the proportion of computation time $p_{comp}$ among the total execution time of the ESB. The A2E algorithm first broadcasts the $p_{comp}$ of current ESB to all other ESBs and thus the highest $p_{comp}$, $p'_{comp}$ can be used as a threshold for future reference. Given a set of CPU frequencies defined for DVFS, we divide the range of possible $p_{comp}$ [0, $p'_{comp}$] into $N_f$ sub-ranges, where $N_f$ is the number of available CPU frequencies. Which sub-range the $p_{comp}$ of an ESB sits determines which CPU frequency to apply for the ESB. Figure 6 contrasts AGGREE and A2E using the same code example shown in Figure 5.

**Example**. Consider a data intensive application with 10 ESBs, among which the highest proportion of computation time within the total execution time is 20%, and there are four gears of CPU frequency available for DVFS. According to Algorithm 1, the range of CPU frequency for adaptively aggressive DVFS scheduling consists of four individual sub-range from 0 to 20%, i.e., [0, 5%), [5%, 10%), [10%, 15%), and [15%, 20%]. If the proportion of computation time for an ESB is within the range of [0, 5%), we set CPU frequency to $f_0$, i.e., the lowest frequency; if the proportion falls into the range of [5%, 10%), we set CPU frequency to $f_1$, i.e., the second lowest frequency, and so on. Consequently for each ESB, we can assign a fitting frequency based on the amount of computation within the ESB.

Although the low-performance trade-off is moderated by A2E, the overhead on employing DVFS increase a bit due to more CPU frequency switches issued by A2E. From Figure 6, we can see that the number of CPU frequency switches approximates the number of ESBs in the `if` branch, since for each ESB, we at least set an appropriate CPU frequency for it once. For the code example shown in Figure 6, we do not need to switch CPU frequency for the ESB within the `else` branch, because we guarantee at the end of the `if` branch CPU frequency is set to high. Overall, the number of CPU frequency switches for A2E approximates $NN_iP_1$, comparable to that for the basic DVFS scheduling $2N_iN_m$, where $N$ is the number of ESBs in the loop, $N_i$ is the number of iterations of the loop, and $N_m$ is the number of *Comm*-ESBs in the loop. Note that different types of workloads do not necessarily appear in a loop, we let $N_i = 1$ when hybrid workloads are not present in a loop, but in a code segment without loops. In this case, the number of CPU frequency switches for A2E dramatically decreases to $NP_1$ that is of the same order of magnitude as that for AGGREE. In other words, the DVFS overhead of A2E and AGGREE are comparable when different types of workloads are present in a code segement without loops.

### E. Speculative DVFS Scheduling for Imbalanced Branches

Speculation is a technique that allows a compiler or a processor to predict the execution of an instruction so that an earlier execution of other instructions depending on the speculated instruction may be enabled. In our case, we speculate the outcome of a branching statement for energy saving purposes. If the application consists of conditional statements with significantly different possibility of occurrence (i.e., imbalanced branches) such as the `if-then-else` construct shown at Lines 2 and 15 in the kernel of an application with different workloads with imbalanced branches as depicted in Figure 5 (a). There are two branches to take where the taken possibility $P_1$ of the `if` branch is much greater than that of the `else` branch $P_2$, which is a real case for the benchmark DT from NPB. As shown in Figures 5 (b) and 6, we can speculatively set CPU frequency to low outside the rarely taken `else` branch inside the loop, and set CPU frequency to high for computation within the `else` branch, as a recovery mechanism used for incorrect speculation, so that the overall performance is not compromised even if the `else` branch is taken empirically.

Speculation can be applied to both AGGREE and A2E to reduce the number of CPU frequency switches for less DVFS overhead. Although in comparison to AGGREE with no speculation and A2E with no speculation, the use of speculation within both approaches slightly increases the number of CPU frequency switches by additional $2N_iP_2$ and $N_iP_2$ times, respectively. Overall, the speculative DVFS scheduling together with AGGREE and A2E effectively reduce the number of CPU frequency switches from $2N_iN_m$ of the basic DVFS scheduling strategy to $2 + 2N_iP_2$ and $NN_iP_1 + N_iP_2$ individually. Following the constraint $P_2 \ll P_1$ due to the imbalanced branches, AGGREE with speculation is more effective on reducing DVFS overhead against A2E with speculation.

### F. Performance Model

Next we model the performance efficiency of the three approaches (Basic DVFS, AGGREE, and A2E) at ESB level for a data intensive application. Since the application consists of different types of ESBs, performance efficiency of each ESB reflects the overall performance efficiency of the application. Table I lists the notation used in the formalization of performance. Given an application with different types of workloads comprised of computation (CPU-bound), communication (network-bound), memory accesses (memory-bound), and disk accesses (disk-bound), we model the performance of the original application without any DVFS scheduling strategies as sum of execution time of different components:

$$T = T_{comp} + T_{comm} + T_{mem} + T_{disk} \quad (1)$$

TABLE I. NOTATION IN PERFORMANCE EFFICIENCY FORMALIZATION.

| | |
|---|---|
| $T$ | Total execution time of the application |
| $T_{comp}$ | Computation time of the application |
| $T_{comm}$ | Communication time of the application |
| $T_{mem}$ | Average memory access time of the application |
| $T_{disk}$ | Average disk access time of the application |
| $O_{comp}$ | Time complexity of computation of the application |
| $f$ | Current CPU operating frequency |
| $f_h$ | A high CPU frequency set by DVFS |
| $f_m$ | A medium CPU frequency set by DVFS adaptively |
| $f_l$ | A low CPU frequency set by DVFS |
| $N_c$ | Number of cores within one node of the cluster |
| $N_F$ | Floating Point Unit of one core divided by 64-bit |
| $T_{DVFS}$ | Time consumed by a DVFS CPU frequency switch |
| $P_1$ | Taken possibility of the likely taken imblanced branch |
| $P_2$ | Taken possibility of the rarely taken imblanced branch |
| $N_i$ | Number of iterations of a loop with hybrid workloads |
| $N$ | Number of ESBs in a hybrid loop/application |
| $N_m$ | Number of *Comm*-ESBs in a hybrid loop/application |

Let us assume the application is executed with the optimal efficiency (100%), $T_{comp}$ can be represented as:

$$T_{comp} = \frac{O_{comp}}{fN_cN_F} \tag{2}$$

As we know, only CPU frequency $f$ in calculating $T_{comp}$ is affected by DVFS, while execution time of operations other than computation is bounded by non-CPU hardware factors such as network bandwith and disk data transfer rate, and is not related to CPU frequency. Therefore we separate the computation accompanying memory and disk accesses from the actual memory and disk accesses for each approach below, where $T'_{mem}$ and $T'_{disk}$ denote the actual memory and disk access time respectively, and the impact of DVFS on execution time is shown by setting different CPU frequencies. Note that each approach employs the same heuristic for energy efficient computation and communication: Keeping the highest CPU performance for computation and applying the lowest CPU performance for communication.

$$\begin{aligned} T_{orig} =& \frac{O_{comp}}{f_hN_cN_F} + T_{comm} + T'_{mem} + \frac{O_{comp\_mem}}{f_hN_cN_F} \\ &+ T'_{disk} + \frac{O_{comp\_disk}}{f_hN_cN_F} \end{aligned} \tag{3}$$

$$\begin{aligned} T_{basic} =& \frac{O_{comp}}{f_hN_cN_F} + T_{comm} + T'_{mem} + \frac{O_{comp\_mem}}{f_hN_cN_F} \\ &+ T'_{disk} + \frac{O_{comp\_disk}}{f_hN_cN_F} + T_{DVFS} \times 2N_iN_m \end{aligned} \tag{4}$$

$$\begin{aligned} T_{aggree} =& \frac{O_{comp}}{f_lN_cN_F} + T_{comm} + T'_{mem} + \frac{O_{comp\_mem}}{f_lN_cN_F} \\ &+ T'_{disk} + \frac{O_{comp\_disk}}{f_lN_cN_F} + T_{DVFS} \times (2 + 2N_iP_2) \end{aligned} \tag{5}$$

$$\begin{aligned} T_{a2e} =& \frac{O_{comp}}{f_hN_cN_F} + T_{comm} + T'_{mem} + \frac{O_{comp\_mem}}{f_mN_cN_F} \\ &+ T'_{disk} + \frac{O_{comp\_disk}}{f'_mN_cN_F} + T_{DVFS} \times NN_iP_1 \end{aligned} \tag{6}$$

Without loss of generality, given a data intensive application with different types of workloads and imblanced branches, since computation only takes a small proportion of the total execution time of the application, we assume $T_{comm} + T'_{mem} + T'_{disk} = n \times T_{comp} = \frac{nO_{comp}}{f_hN_cN_F}$, where $n > 1$. The last added items in Equations 4, 5, 6 are the overhead on employing DVFS. We know $P_2$ approximates to 0 since this branch is rarely taken, so the DVFS overhead is negligible for AGGREE. Additionally, from Table III we can see that in our experimental platform $f_h \approx 3f_l$ if we adopt Gear 0 as $f_h$ and Gear 3 as $f_l$ for AGGREE, and we assume $f_m = mf_l$ and $f'_m = m'f_l$. Thus we obtain the simplified formulae of performance for the three approaches as:

TABLE II. NOTATION IN ENERGY EFFICIENCY FORMALIZATION.

| | |
|---|---|
| $E_{sys}$ | Total energy consumption of the whole cluster |
| $E_{node}$ | Total energy consumption of all components in a node |
| $P_{node}$ | Total power consumption of all components in a node |
| $P_{CPU\_d}$ | CPU dynamic power consumption in the busy state |
| $P_{CPU\_s}$ | CPU static/leakage power consumption in any states |
| $P_{other}$ | Power consumption of components other than CPU |
| $A$ | Percentage of active gates in the CMOS-based chip |
| $C$ | Total capacitive load in the CMOS-based chip |
| $V$ | Current CPU supply voltage |
| $V_h$ | A high supply voltage set using DVFS |
| $V_l$ | A low supply voltage set using DVFS |
| $n$ | Time ratio between non-computation and computation |

TABLE III. FREQUENCY-VOLTAGE PAIRS FOR THE AMD OPTERON 2380 PROCESSOR.

| Gear | Frequency (GHz) | Voltage (V) |
|---|---|---|
| 0 | 2.5 | 1.35 |
| 1 | 1.8 | 1.2 |
| 2 | 1.3 | 1.1 |
| 3 | 0.8 | 1.025 |

$$T_{orig} \approx \frac{(n+1)O_{comp} + O_{comp\_mem} + O_{comp\_disk}}{3f_lN_cN_F} \tag{7}$$

$$\begin{aligned} T_{basic} \approx& \frac{(n+1)O_{comp} + O_{comp\_mem} + O_{comp\_disk}}{3f_lN_cN_F} \\ &+ T_{DVFS} \times 2N_iN_m \end{aligned} \tag{8}$$

$$\begin{aligned} T_{aggree} \approx& \frac{(\frac{n}{3}+1)O_{comp} + O_{comp\_mem} + O_{comp\_disk}}{f_lN_cN_F} \\ &+ T_{DVFS} \times (2 + 2N_iP_2) \end{aligned} \tag{9}$$

$$\begin{aligned} T_{a2e} =& \frac{(\frac{n+1}{3})O_{comp} + \frac{1}{m}O_{comp\_mem} + \frac{1}{m'}O_{comp\_disk}}{f_lN_cN_F} \\ &+ T_{DVFS} \times (NN_iP_1) \end{aligned} \tag{10}$$

From the comparison between Equations 7, 8, 9, and 10, we can see that against the original application without any DVFS strategies, the basic DVFS scheduling strategy only results in performance loss due to additional DVFS overhead, while both AGGREE and A2E incur performance loss from reducing CPU performance during computation. Compared to AGGREE, performance loss from A2E is moderated, since each coefficient of computation time complexity of A2E is smaller than that of AGGREE. Moreover, A2E suffers from DVFS overhead comparable to the basic DVFS scheduling strategy, while AGGREE has the minimal overhead on using DVFS due to the constraint $P_2 \ll P_1$ for imbalanced branches.

### G. Energy Model and Energy Efficiency Analysis

We next formalize energy saving opportunities provided by the three energy efficient approaches individually using the notation in Table II. Within a given time interval ($t_1$, $t_2$), the total energy costs of a distributed-memory computing system consisting of multiple computing nodes can be formulated as below, where we denote the execution time as $T = t_2 - t_1$ and the nodal average power consumption as $\overline{P_{node}}$:

$$E_{sys} = \sum_{1}^{\#nodes} E_{node} = \sum_{1}^{\#nodes} \int_{t_1}^{t_2} P_{node} dt = \sum_{1}^{\#nodes} \overline{P_{node}} \times T \quad (11)$$

Assuming each node in the computing system has the same hardware configuration and local energy efficiency results in global energy efficiency according to Equation 11, we only consider nodal energy consumption in the later discussion. Generally, we break down nodal power consumption as:

$$P_{node} = P_{CPU\_d} + P_{CPU\_s} + P_{other};\ P_{CPU\_d} \approx ACfV^2 \quad (12)$$

In (12), we categorize the nodel power consumption by power consumption of CPU and other components. By substituting $P_{CPU\_d}$, we obtain the ultimate nodal power consumption formula with DVFS-dependent parameters $f$ and $V$ as:

$$P_{node} \approx ACfV^2 + P_{CPU\_s} + P_{other} \quad (13)$$

In our case, $P_{CPU\_s}$ and $P_{other}$ barely change during the execution and thus we denote $P_{CPU\_s} + P_{other}$ as a constant $P_c$. From Equation 9, we know that the DVFS overhead of AGGREE is negligible due to the presence of $P_2$. Following the constraints of Equations 11, 12, and 13, we can calculate energy costs of running a data intensive application with different DVFS scheduling strategies respectively. Further, we model the energy savings achieved by AGGREE and A2E in contrast to the original application individually as below:

$$\begin{aligned} \Delta E_{aggree} &= E_{node}^{orig} - E_{node}^{aggree} = \overline{P_{node}^{orig}} \times T_{orig} - \overline{P_{node}^{aggree}} \times T_{aggree} \\ &\approx (ACf_hV_h^2 + P_c)T_{orig} - (ACf_lV_l^2 + P_c)T_{aggree} \end{aligned} \quad (14)$$

$$\begin{aligned} \Delta E_{a2e} &= E_{node}^{orig} - E_{node}^{a2e} = \overline{P_{node}^{orig}} \times T_{orig} - \overline{P_{node}^{a2e}} \times T_{a2e} \\ &\approx (ACf_hV_h^2 + P_c)T_{orig} - (\overline{ACfV^2} + P_c)T_{a2e} \end{aligned} \quad (15)$$

From Equations 14 and 15, we observe that there exists a performance-energy trade-off that should be considered to determine the optimal CPU frequency to employ in different requirements. In our scenario, achieving the maximal energy savings with minor performance loss is the goal. For evaluating if the energy efficiency achieved and the performance degradation incurred are balanced, we adopt an integrated metric to quantify the energy-performance efficiency: Energy-Delay Product (EDP), a widely used metric to weigh the comprehensive effects of energy and performance for a given application under different configurations [23]. Therefore, we leverage the EDP metric and its variant ED2P to evaluate among the three energy efficient approaches, which one is able to achieve the optimal energy-performance efficiency (the smaller value, the better efficiency) for data intensive applications. Details of the implementation and evaluation of all three energy efficient approaches are illustrated next.

TABLE IV. BENCHMARK DETAILS.

| Benchmark | Source | Test Case | Category |
|---|---|---|---|
| DT | NPB | Class B | Memory Access Intensive and Imbalanced Branches |
| MG | NPB | Class C | Memory and Disk Access Intensive |
| SPhot | ASC | Track 4000 particles | Memory Access Intensive |
| MPIBZIP2 | bzip2 | Compress a 0.77GB file | Disk Access Intensive |
| cp_MPI | Linux | Copy a file of 54.4MB | Disk Access Intensive |

## V. IMPLEMENTATION AND EVALUATION

We have implemented all three energy efficient DVFS scheduling strategies and evaluated their effectiveness towards five high performance data intensive applications with different dominant workloads such as memory and disk accesses with imbalanced branches. Instead of assigning appropriate CPU frequencies to an ESB with differet types of workloads in a fine-grained fashion, we aggressively schedule CPU frequency to an intermediate value for memory and disk accesses mixed with minor computation adaptively according to the proportion of computation time among the total execution time, in order to achieve considerable energy savings at the cost of minor performance loss. As for imbalanced branches, we adopt speculative DVFS scheduling to reduce the number of CPU frequency switches to minimize the overhead on employing DVFS. The DVFS technique in our approach through modifying CPU frequency configuration files dynamically at system level enables us to scale CPU voltage and frequency up and down if necessary for energy efficiency. Benchmarks used consist of various sources of memory and disk access intensive programs with imbalanced branches, such as DT and MG from NPB and ASC benchmark suites [22] [24], an MPI version of the high-quality data compressor bzip2 [25], and an self-written MPI version of the Linux standard file copy command cp [26]. Table IV shows the details of benchmarks.

### A. Experimental Setup

We applied the three DVFS scheduling approaches individually to the five benchmarks to assess their effectiveness of energy savings and performance loss trade-off. Experiments were performed on a computing cluster with an Ethernet switch consisting of 8 computing nodes with two Quad-core 2.5 GHz AMD Opteron 2380 processors (totalling 64 cores) and 8 GB RAM running 64-bit Linux kernel 2.6.32, The power-aware and DVFS-enabled cluster was equipped with power sensors and meters for energy measurement. In our experiments, time was measured using the MPI_Wtime() routine. Energy consumption was measured using PowerPack [23], a comprehensive software and hardware framework for energy profiling and analysis of high performance systems and applications. The range of CPU frequency on HPCL was {0.8, 1.3, 1.8, 2.5} GHz. PowerPack was deployed and running at a meter node within the cluster to collect energy costs on all involved components such as CPU, memory, disk, motherboard, etc. on all 8 computing nodes of the cluster. The collected energy information was recorded into a log file in the local disk and accessed after execution of these benchmarks.

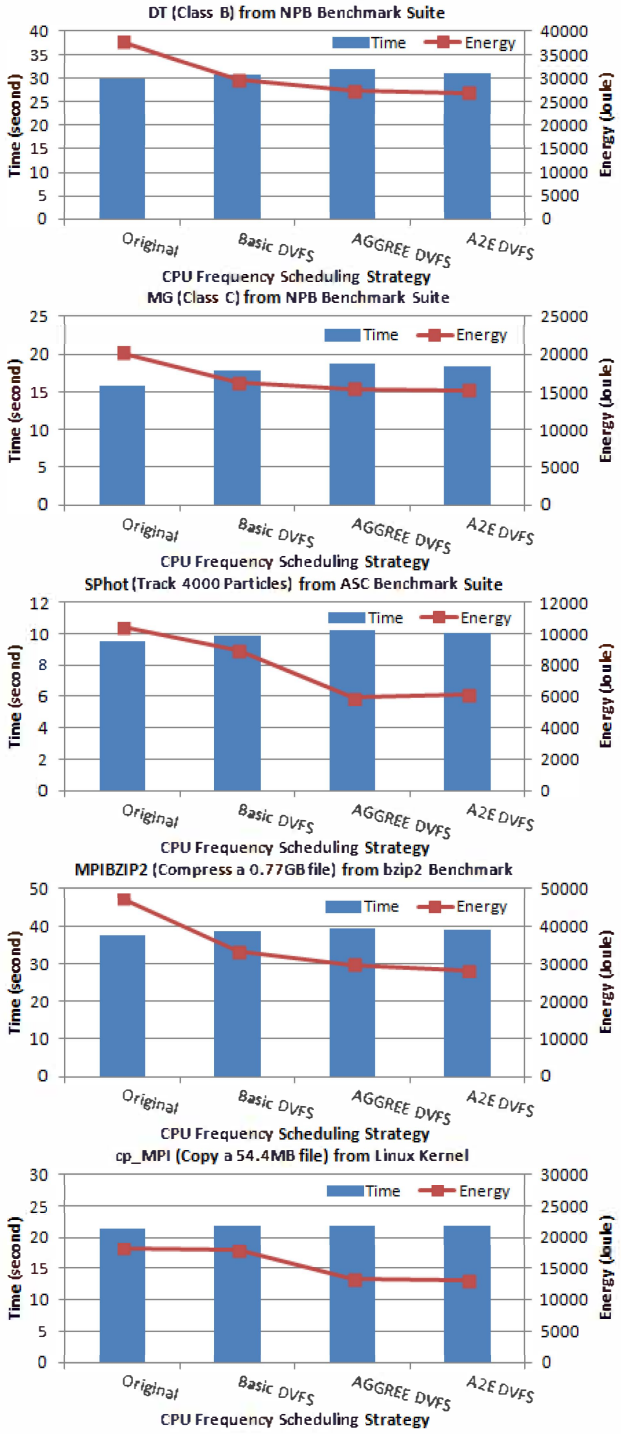


Fig. 7. Performance Loss and Energy Savings on a Cluster with 8 Nodes, 64 Cores of {0.8, 1.3, 1.8, 2.5} GHz CPU frequencies, and 8 GB Memory/Node.

### B. Performance Degradation

All three DVFS scheduling approaches improve energy efficiency for data intensive applications at the cost of minor performance loss as shown in Figure 7, where the $x$ axis label Original denotes the original application without any DVFS scheduling strategies, and Basic DVFS, AGGREE DVFS, and A2E represent the basic, AGGREE, and A2E DVFS scheduling strategies introduced in the last section, respectively. We can see that in general A2E incurs similar performance loss as the basic DVFS scheduling strategy compared to the original no-DVFS executions: 6.2% and 4.7% on average, respectively, while AGGREE degrades performance more (8.1% on average) due to aggressively lowering down CPU performance regardless of minor computation within the data intensive application.

The overhead on employing DVFS in the basic DVFS scheduling strategy primarily results from two factors: (a) The additional time spent on modifying CPU frequency configuration files dynamically at system level and (b) CPU frequency transition latency. Thus the number of CPU frequency switches by DVFS determines the DVFS overhead. Some application such as MG incurs up to 13.0% performance loss due to applying DVFS, since there exist a great amount of alternate *Comm*-ESBs and *Comp*-ESBs as shown in Figure 5 (a), which requires a large amount of CPU frequency switches by DVFS as well. The communication time for some application like cp_MPI is negligible and the amount of *Comm*-ESBs is limited. Therefore constrained by both factors, the overhead on employing DVFS for cp_MPI is also negligible (1.5%).

As discussed before, performance loss from AGGREE is attributed to low performance of the small proportion of computation mixed with memory and disk accesses. According to Equation 5, reducing CPU frequency aggressively results in longer execution time for the CPU-bound computation and thus incurs overall performance degradation, although performance of memory and disk accesses is barely affected. Since the ratio between computation and non-computation is significantly low in memory and disk access intensive applications, the impact of performance loss from computation is limited on the total execution time. A2E further decreases the performance loss by adaptively scheduling an appropriate CPU frequency to an ESB according to the computation time proportion instead of always setting the lowest CPU frequency. On the other hand, AGGREE and A2E successfully reduce the number of CPU frequency switches by applying DVFS outside of a loop of ESBs and inside a rarely taken branch, respectively. The DVFS overhead is reduced accordingly compared to the basic DVFS scheduling strategy where there exist a larger number of CPU frequency switches due to fine-grained DVFS scheduling before and after each *Comm*-ESB within the loop. Consequently, with less DVFS overhead, AGGREE and A2E only suffer from 3.4% and 1.5% more performance loss than that of the basic DVFS scheduling strategy, respectively.

### C. Energy Savings for Memory Access Intensive Applications

Figure 7 also reflects energy efficiency achieved by the three approaches. Compared to the basic and AGGREE DVFS scheduling strategies, A2E is able to achieve more energy savings, since energy saving opportunities from memory and disk accesses that the basic DVFS scheduling strategy fails to leverage are exploited by AGGREE and A2E as depicted in Figures 5 (b) and 6 (b) respectively, and further moderation of low-performance trade-off is performed by A2E against AGGREE. Specifically, considering energy consumption of the original executions as the baseline, 32.6% on average energy savings are fulfilled by A2E, in contrast to 17.3% and 31.7% energy savings on average achieved by the basic and AGGREE DVFS scheduling strategies individually.

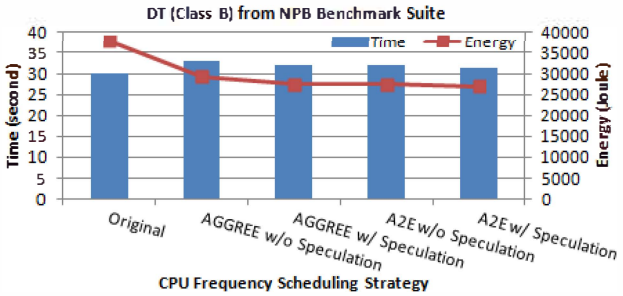


Fig. 8. Performance and Energy Efficiency upon Employing Speculation in AGGREE and A2E for the DT Benchmark with Imbalanced Branches.

The most energy savings 43.4% AGGREE achieves is for the memory access intensive application SPhot, while A2E manages to achieve less energy savings 40.9%. We applied AGGREE and A2E to a code segment within SPhot, where a great amount of memory accesses mixed with calculating array indices before accessing corresponding memory locations are present in a double-loop. The basic DVFS scheduling strategy only obtains 13.9% energy savings, since it fails to handle *Mem*-ESBs accompanied by computation but only saves energy for *Comm*-ESBs. With AGGREE employed, performance of SPhot is degraded by 7.2% due to low performance of memory address calculation interleaved in memory accesses as a consequence of aggressively scaling down CPU frequency. Performance loss is moderated by A2E to 4.9% at the cost of less energy savings, since the *Mem*-ESBs of SPhot have similar proportion of computation time and thus most CPU frequencies adaptively assigned are close to the highest one.

### D. Energy Savings for Disk Access Intensive Applications

Besides memory access intensive applications, A2E performs better than the other two approaches in gaining energy efficiency for disk access intensive applications. Regarding the disk access dominant application cp_MPI, the basic DVFS scheduling strategy saves a limited amount of energy (2.1%) since the communication time is significant low compared to the disk access time. AGGREE and A2E can obtain more energy savings for this type of applications, since aggressively reducing CPU frequency barely affects performance of the application. As for cp_MPI, most execution time is spent on non-CPU-bound opertions, disk accesses, whose execution time is constrained by non-CPU hardware factors such as average seek time and disk data transfer rate. Low CPU performance brings in considerable energy savings from CPU during data waiting time without significant performance loss as a whole. Another disk access intensive application MPIBZIP2 also benefits from the moderation of low-performance loss by A2E with 40.5% energy savings achieved compared to 37.1% from AGGREE.

Note that although similar percentage of energy savings are fulfilled for memory access intensive and disk access intensive applications, performance degradation for employing aggressive DVFS scheduling strategies like AGGREE and A2E towards the two types of applications differ: Despite MG, an application with comparable memory and disk accesses, memory access intensive applications (DT and SPhot) suffer from average performance loss of 7.2% for AGGREE and 4.7% for A2E, while disk access intensive applications (MPIBZIP2 and cp_MPI) only sacrifice minor performance loss on average of

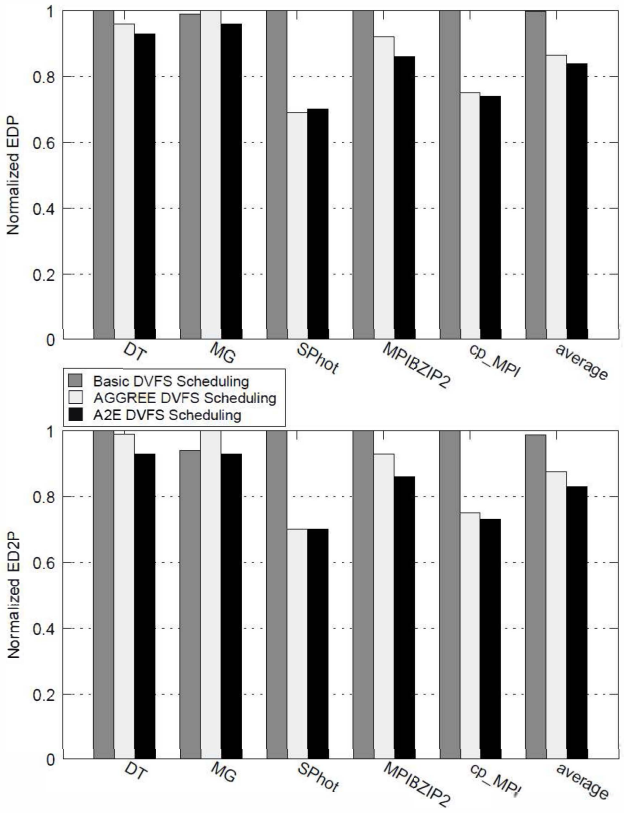


Fig. 9. Energy-Performance Efficiency Trade-off in Terms of EDP and ED2P.

3.4% for AGGREE and 2.7% for A2E. This is attributed to two causes: (a) Memory access time is much smaller than disk access time (typically with a ratio of the order of magnitude $1/10^6$) and thus is closer to CPU clock cycles; (b) The amount of computation mixed with memory accesses is generally more than that with disk accesses. Both reasons make the impact of CPU performance degradation on the total execution time of memory access intensive applications greater than that of disk access intensive applications. It is notable that moderating performance loss from A2E shrinks the gap.

### E. Energy Savings for Imbalanced Branches

AGGREE and A2E adopt speculation to further gain energy efficiency for code with imbalanced branches by reducing the DVFS overhead. DT is a memory intensive graph application where a great amount of imbalanced branches exist. Figure 8 shows energy consumption and execution time of DT using AGGREE and A2E with and without speculation individually. We can see that employing speculation within AGGREE and A2E mitigates performance degradation and thus saves energy: Performance loss from AGGREE drops from 10.8% to 7.1%, while energy savings increase from 22.7% to 27.4%; performance loss from A2E drops from 6.8% to 4.4%, while energy savings increase from 27.3% to 28.6%. The effectiveness of speculation for saving time and energy results from aggressively reducing CPU frequency for the frequently taken branch while keeping CPU frequency high for computation within the rarely taken branch as the recovery mechanism used for incorrect speculation, as shown in Figures 5 (b) and 6. Note that A2E is empirically less effective than

AGGREE in reducing the DVFS overhead upon the use of speculation, which is consistent with the performance loss from employing DVFS calculated formally in section 4.

### F. Energy and Performance Efficiency Trade-off

From Equations 14 and 15, we observe there exists an energy-performance efficiency trade-off for AGGREE and A2E. In general, moderating CPU performance degradation by adaptively scheduling an intermediate rather than always the lowest CPU frequency for a *Mem*-ESB or a *Disk*-ESB deceases performance loss at the cost of higher average power, since power is proportional to CPU frequency and voltage. Variation of performance and energy efficiency at different operating points can be quantified by an integrated metric that both impacts of performance and energy are considered. We adopt the EDP (Energy-Delay Product) metric and its variant ED2P (Energy-Delay-Squared Product) to evaluate the balance between energy and performance efficiency for the three energy saving approaches, as presented in Figure 9.

Since a smaller value in the EDP and ED2P metrics represents higher energy and performance efficiency as a whole, we can see that for data intensive applications, the basic DVFS scheduling strategy is not the optimal approach since it fails to exploit the energy saving opportunities present in the operations other than communication. Except that for SPhot, A2E and AGGREE have similar EDP and ED2P values, A2E is superior to AGGREE for all other applications in terms of the balance of energy-performance efficiency. The average values of EDP and ED2P for A2E and AGGREE over all five benchmark consolidate this observation.

## VI. Conclusions

Driven by the growing energy concerns, DVFS techniques have been widely applied to improve energy efficiency for high performance applications on distributed-memory computing systems nowadays. Energy saving opportunities from slack in terms of load imbalance, network latency, communication dalay, memory and disk access stalls, etc. are exploited to save energy through scaling up and down CPU voltage and frequency via DVFS, since peak CPU performance is not necessary during the slack. We propose an adaptively aggressive energy efficient DVFS scheduling strategy (A2E) for data intensive applications such as memory and disk access intensive applications with imbalanced branches. Instead of assigning CPU frequency in a fine-grained fashion towards an Energy Saving Block (ESB) with different types of workloads, A2E adaptively schedules an appropriate CPU frequency for the hybrid ESB aggressively as a whole and reduces the overhead on employing DVFS via speculation to save energy with minor performance loss. The experimental results indicate the effectiveness of A2E for saving energy of running target applications with minor performance loss.

## Acknowledgments

The authors would like to thank the HPCL Lab at the Department of Mathematics, Statistics, and Computer Science of Marquette University for providing the distributed-memory computing cluster with the energy/power profiler PowerPack.

This research is partly supported by US National Science Foundation, under the grants #CNS-1118043, #CNS-1116691, #CNS-1304969, #CNS-1305359, and #CNS-1305382.

## References

[1] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proc. SC*, 2005, p. 34.

[2] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Proc. CCGRID*, 2007, pp. 541–548.

[3] L. Tan, M. Feng, and R. Gupta, "Lightweight fault detection in parallelized programs," in *CGO*, 2013, pp. 1–11.

[4] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. OSDI*, 1994, p. 2.

[5] *CPUFreq*. https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling.

[6] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: Understanding the runtime effects of frequency scaling," in *Proc. ICS*, 2002, pp. 35–44.

[7] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing power with performance constraints for parallel sparse applications," in *Proc. IPDPS*, 2005, pp. 1–8.

[8] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs," in *Proc. SC*, 2005, p. 33.

[9] C.-H. Hsu and W.-C. Feng, "A power-aware run-time system for high-performance computing," in *Proc. SC*, 2005, p. 1.

[10] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," in *Proc. PPoPP*, 2005, pp. 164–173.

[11] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh, "Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster," in *Proc. PPoPP*, 2006, pp. 230–238.

[12] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster," in *Proc. CLUSTER*, 2006, pp. 1–10.

[13] R. Ge, X. Feng, W.-C. Feng, and K. W. Cameron, "CPU MISER: A performance-directed, run-time system for power-aware clusters," in *Proc. ICPP*, 2007, p. 18.

[14] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Proc. SC*, 2007, pp. 1–9.

[15] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS practical for complex HPC applications," in *Proc. ICS*, 2009, pp. 460–469.

[16] L. Tan and Z. Chen, "Beating critical path approach: Algorithmic Cholesky factorization energy saving," in *submission*, 2013.

[17] L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, and D. Li, "Improving performance and energy efficiency of matrix multiplication via pipeline broadcast," in *Proc. CLUSTER*, 2013.

[18] ——, "DAEMON: Distributed adaptive energy-efficient matrix-multiplication," in *submission*, 2013.

[19] D. Li, B. R. de Supinski, M. Dolz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy efficient resource management of hybrid programming models," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 144–157, Jan. 2013.

[20] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and X. Lin, "Power-efficient time-sensitive mapping in heterogeneous systems," in *Proc. PACT*, 2012, pp. 23–32.

[21] Y. Luo, V. Packirisamy, W.-C. Hsu, and A. Zhai, "Energy efficient speculative threads: Dynamic thread allocation in same-ISA heterogeneous multicore systems," in *Proc. PACT*, 2010, pp. 453–464.

[22] *NAS Parallel Benchmarks*. http://www.nas.nasa.gov/publications/npb.html.

[23] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, May 2010.

[24] *ASC Sequoia Benchmark Codes*. https://asc.llnl.gov/sequoia/benchmarks/.

[25] *Parallel MPI BZIP2 (MPIBZIP2)*. http://compression.ca/mpibzip2/.

[26] *Linux File Copy Command*. http://www.linux.org/article/view/useful-commands-the-cp-command.