Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 6, June 2018, pp. 1201-1212
DOI: 10.23940/ijpe.18.06.p11.12011212

Reducing Energy Cost of Multi-Threaded Programs on NUMA Architectures

Hao Fang[a], Liang Zhu[b,*], and Xiangyu Li[a]

[a]*School of Computer Science, Wuhan Donghu University, Wuhan, 430212, China*
[b]*China Ship Development and Design Center, Wuhan, 430064, China*

Abstract

Many recent data center servers are built with NUMA (Non-Uniform Memory Access) characteristics. Accessing remote memory generally takes longer time than accessing local memory. There are a lot of research works that discuss the performance improvement of NUMA multi-core systems. However, rare research work considers reducing the energy cost of NUMA multi-core systems. This work studies reducing energy cost of multi-threaded programs on NUMA architectures using DVFS (Dynamic Voltage and Frequency Scaling) adjustment strategy. We consider three factors of the multi-threaded programs which influence the energy saved by our DVFS adjustment strategy. These three factors are: (1) the memory access intensity of parallel programs; (2) the proportion of remote memory access; (3) the ratio between remote and local memory access latency. In addition, we propose two DVFS adjustment strategies to save the energy cost of multi-threaded programs. The energy-saving effect of these two DVFS adjustment strategies is influenced by these three factors. Two DVFS adjustment strategies can save maximally 20% and 39.2% of total energy when considering one factor and 33.3%, 48.1% of total energy when considering two factors, respectively.

Keywords: multi-threaded programs; Non-Uniform Memory Access (NUMA); Dynamic Voltage and Frequency Scaling (DVFS); remote memory access; critical threads

(Submitted on March 6, 2018; Revised on April 12, 2018; Accepted on May 26, 2018)

1. Introduction

Multiple memory controllers can improve the scalability of multi-core systems and therefore lead to the characteristics of NUMA (Non-Uniform Memory Access). At present, cloud servers are composed of certain number of processor nodes which have their own memory controllers. Each node contains a multi-core processor, a main memory and a memory controller. These nodes are combined by high speed inter-connect links so as to form a cache-coherent memory system. All cores of CPUs can access memory in all nodes transparently by means of shared memory coherence protocol. Accessing the program data which is allocated on the local node only needs to go through a local in-memory controller. Accessing the program data which is allocated on remote nodes has to go through the inter-connect link and a remote memory controller. The time of accessing remote memory (310 cycles) is much longer than the time of accessing local memory (190 cycles), which is as shown in Figure 1. This property is described as NUMA.

High energy consumption is one of the major challenges to be solved in designing the NUMA multicore systems and other systems [1,25]. The memory access time imbalance among cores in different nodes is one source of energy inefficiency. For example, in a fork-join parallel programming model (e.g., OpenMP parallel for regions [3,21]), the total work is evenly distributed among all threads. All threads can arrive at synchronization points (e.g., barriers) simultaneously if the memory access time is nearly the same for all threads. However, due to the characteristics of NUMA multi-core systems, some threads have to access the data, which is allocated on remote node; thus, those have longer memory access time. As shown in Figure 2, the initial thread T1 initializes all program data on local node. Thread T2, T3 and T4 are also located on the local node. Therefore, they all have low memory access time due to the fact that they only need go through

* Corresponding author.
E-mail address: lemonsprite@qq.com

the local memory controller to access local data. Thread T5 is located on the remote node. As a consequence, T5 has very high memory access time because it has to go across the inter-connect link and remote memory controller to access remote data. Thread T1, T2, T3 and T4 have to busy wait for Thread T5 arriving at synchronization points, even if they finish their allocated work much earlier than Thread T5. During the busy waiting phase, they cost the same power when they are doing useful work.

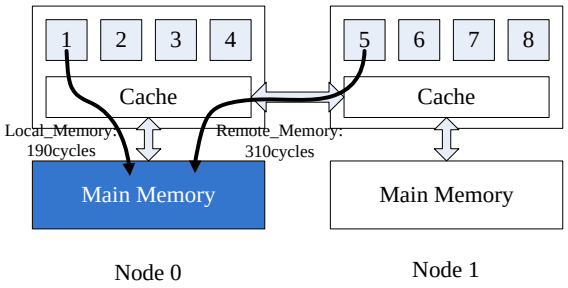


Figure 1. The memory access latencies of local and remote memory on NUMA architecture

There are two ways to save energy on NUMA multicore systems. Firstly, we can turn off the core which has nothing useful to do but busy wait. During the turn-off phase, the CPU core only consumes the leakage power, which is much smaller than the dynamic power of the CPU core. In our paper, this way is called before-hand DVFS adjustment strategy, due to the fact that this strategy can be used as long as the CPU core has nothing useful to do [2]. Secondly, the DVFS (Dynamic Voltage and Frequency Scaling) technology can be used to save a large amount of energy consumed by the CPU core by adjusting the frequency of the CPU core. Increasing or decreasing the voltage of the CPU core can speed up or slow down the frequency of the CPU core. The performance of the CPU core is in proportional to the frequency of the CPU core. We can cut down the runtime of the thread by speeding up the frequency of the CPU core and prolong the runtime of the thread by slowing down the frequency of the CPU core. The power cost of the CPU core is the square of the frequency of the CPU core. Slowing down a little of the frequency of the CPU core can save a large amount of energy due to the fact that the wasted energy is the cube of the frequency of the CPU core [15,17]. In our paper, this technology is called back-hand DVFS adjustment strategy due to the fact that this strategy can be used only after the essential runtime information. For example, which threads will finish their work earlier, is acquired by this strategy.

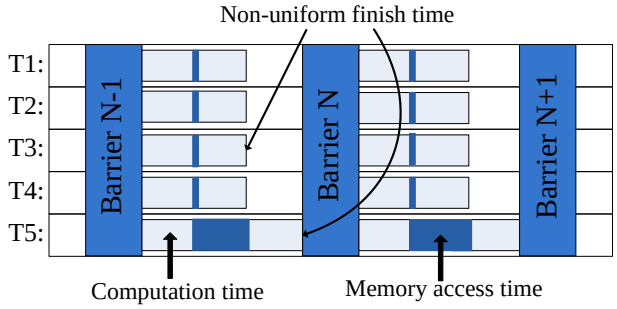


Figure 2. Runtime imbalance caused by the non-uniform memory access time

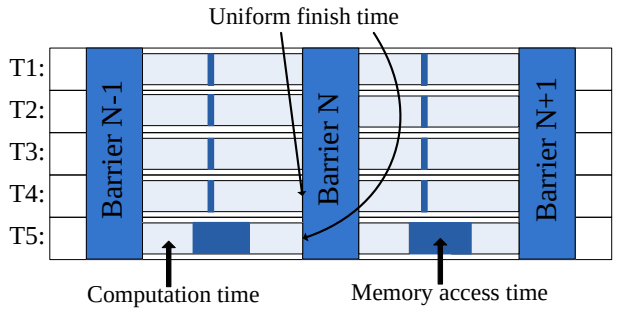


Figure 3. Runtime balance by using of the DVFS adjustment strategy

In Figure 2, Thread T5 is called the critical thread due to the fact that it lags down the overall performance of the multi-threaded program and all the other threads have to wait it to finish. The other threads T1, T2, T3 and T4 are called non-critical threads because their runtime is shorter than the runtime of T5. Extending the runtime of threads T1, T2, T3 and T4

does not influence the finish time of the multi-threaded program as long as the runtime of threads T1, T2, T3 and T4 is not longer than the runtime of Thread T5. Extending the runtime of these threads means that we can slow down the frequency of the CPU cores that run these threads [16]. In the ideal situation, we can maximally slow down the frequency of the CPU cores to make the critical thread and non-critical threads arriving at synchronization points simultaneously, which is as shown in Figure 3. In this way, we can save the maximally amount of energy.

The rest of this paper is structured as follows: Section 2 introduces the motivation of our work. Section 3 describes the mathematical analysis model [26]. Section 4 presents the simulation results. Related work and conclusions are given in Section 5 and Section 6, respectively.

2. Motivation

The NUMA multicore machine is shown in Figure 1. The machine consists of 2 NUMA nodes. There are 4 cores on each NUMA node. 2 NUMA nodes are connected by the inter-connect link. Each NUMA node has its own memory, which is connected directly to it. Furthermore, it can access the memory of other NUMA nodes through the inter-connect link. The cores on the NUMA node access the memory of its own node only through the local memory controller. The cores which need to access the memory of other NUMA nodes have to cross the inter-connect link and the remote memory controller. The cores have low memory access latency when they access local memory; they have high memory access latency when they access remote memory.

We use a case to illustrate the motivation of our work. A multi-threaded program runs on the NUMA machine, as shown in Figure 1. At the beginning of the execution phase of the multi-threaded program, the initial thread of the multi-threaded program initializes all the program data on the NUMA node on which the initial thread allocates. This is due to the fact that Linux operating systems always allocate the data on the memory of the NUMA node on which the thread that first touches the data allocates. In this case, the program data is allocated on node 0, as shown in Figure 1. The cores which access the data of local memory spend less time, as compared to the cores which access the data of remote memory. According to the measured data from [11], Core 1 needs 190 cycles to access the local memory data of Node 0; however, Core 5 needs 310 cycles to access the remote memory data of Node 0 in Intel Nehalem architectures, as shown in Figure 1.

We use the R_{i_local} and R_{i_remote} to represent the ratio of local and remote memory access of the thread on Core *i*, respectively. We use the L_{local} and L_{remote} to represent the local and remote memory access latency, respectively. We use N_{memory} to represent the total number of memory access instructions of the thread during the parallel execution phase. The following equation is used to compute the memory access time of the thread on Core *i*:

$$T_{memory} = N_{memory} \times R_{i\_local} \times L_{local} + N_{memory} \times R_{i\_remote} \times L_{remote} \quad (1)$$

Due to the fact that the parallel program programmers always allocate equal number of work for each thread to satisfy the load balancing condition of the operating system and threads always execute the same piece of code [6,12], we can conclude that N_{memory} is highly similar for all the threads of one multi-threaded program. Some researches support the assumption that all threads of one multi-threaded program have very similar characteristics compared with each other [14,28]. Therefore, we assume the N_{memory} is equal to all threads of one multi-threaded program. L_{local} and L_{remote} are fixed values which are determined by the system hardware characteristics and L_{remote} is bigger than L_{local}. According to equation (1), T_{memory} is only influenced by R_{i_local} and R_{i_remote}. Due to the fact that the program data is allocated on Node 0, the R_{local} of the thread on Core 1 is higher than the R_{local} of the thread on Core 5 and the R_{remote} of the thread on Core 1 is lower than the R_{remote} of the thread on Core 5. Therefore, T_{memory} of the thread on Core 1 is smaller than T_{memory} of the thread on Core 5. From the above, the total execution time of the thread on Core 1 is smaller than that on Core 5. The thread on Core 1 will get earlier than the thread on Core 5 at synchronization points (e.g., the barriers) of the multi-threaded program.

We can use the DVFS method to slow down those threads that arrive earlier at the synchronization points. The speed of cores is proportional to clock frequency. Using lower clock frequency of Core 1 makes the thread on Core 1 run slower. Choosing the right low clock frequency of Core 1 can make the thread on Core 1 arrive almost simultaneously at synchronization points with the thread on Core 5. The power cost of the core is the cube of the clock frequency of the core. Therefore, reducing the clock frequency of cores can save a large amount of energy during the execution phase of the multi-threaded program.

We discovered one example of this circumstance in real-world environments. We run the *streamcluster* benchmark from the Parsec benchmark [5] on the NUMA machine, as shown in Figure 1. We use the numatop tool [27] to record the

performance data during the execution phase of *streamcluster*. The recorded data is shown in Table 1. RMA(K) and LMA(K) mean the number of remote and local memory access (unit: 1000), respectively. RPI(K) and LPI(K) mean the RMA and LMA normalized by 1000 instructions, respectively.

Table 1. The recorded performance data of *streamcluster*

NODE	RPI(K)	LPI(K)	RMA(K)	LMA(K)	RMA/LMA	CPI	CPU%
0	0.0	11.6	360.5	500710.2	0.0	1.79	74.1
1	11.6	0.0	498266.9	1037.8	480.1	2.30	94.8

From Table 1, we can conclude that the program data is allocated on Node 0 because almost all memory accesses on Node 0 are local memory accesses while almost all memory accesses on Node 1 are remote memory accesses. The LMA(K) of Node 0 is nearly equal to the RMA(K) of Node 1. We can also discover that both the CPI and CPU% of Node 0 is about 22% lower than that of Node 1. In this example, we can find out the enormous amount of remote memory access of Node 1 makes the threads on Node 1 arrive later at the synchronization points [22]. On the contrary, the memory accesses of threads on Node 0 are almost local. The threads on Node 0 arrive earlier at the synchronization points and make the cores of Node 0 have nothing to do but wait for the threads on Node 1 arriving at the synchronization points. Reducing clock frequency of the cores on Node 0 can make the threads on both Node 0 and Node 1 arrive at the synchronization points simultaneously. We can make the clock frequency of the cores on Node 0 22% slow down and not influence the overall performance of *streamcluster*. In this way, we save $1-(1-22\%)^3=52.5\%$ of total energy of Node 0 during the execution period of *streamcluster* program.

3. Analysis Model

A two-node NUMA system is used in our analysis model. The critical thread and non-critical thread run on the local and remote node, respectively. We use a model that considers the execution time of threads, the power cost of cores and the DVFS adjustment strategies of cores. The execution time of threads includes the memory access time and the compute time. The power cost of cores contains the dynamic power cost of cores and the static power cost of cores. The DVFS adjustment strategies of cores comprises the before-hand adjustment strategy and the back-hand adjustment strategy.

3.1. Considering the Execution Time of Threads

The execution time of threads is T_{total}. Assume the memory access time T_{memory} and the compute time $T_{compute}$ occupies α and $1-\alpha$ percent of T_0, respectively. The equation is given below:

$$T_{total} = T_{memory} + T_{compute} = \alpha T_{total} + (1-\alpha)T_{total} \tag{2}$$

The memory access time can be divided as the local memory access time T_{m_local} and the remote memory access time T_{m_remote}:

$$T_{memory} = T_{m\_local} + T_{m\_remote} \tag{3}$$

We define T_{total} is the total executed instruction number of the thread when it runs between two synchronization points. Assume the local and remote memory access instruction number is N_{local} and N_{remote} respectively. The local and remote memory access latency is L_{local} and L_{remote} respectively. N_{local} and N_{remote} accounts for R_{local} and R_{remote} percent of the total instruction number N_{total}. We have the equations below:

$$T_{m\_local} = R_{local} * N_{total} * L_{local} \tag{4}$$

$$T_{m\_remote} = R_{remote} * N_{total} * L_{remote} \tag{5}$$

And,

$$R_{local} + R_{remote} = 1 \tag{6}$$

We define the ratio between L_{remote} and L_{local} is N_{factor}, which describes how many times larger is the remote memory access latency than the local memory access latency on the NUMA system.

$$\begin{aligned} T_{memory} &= T_{m\_local} + T_{m\_remote} \\ &= R_{local} * N_{total} * L_{local} + R_{remote} * N_{total} * L_{local} * N_{factor} \\ &= N_{total} * L_{local} * (R_{local} + R_{remote} * N_{factor}) \end{aligned} \tag{7}$$

N_{total}, L_{local} and N_{factor} is the same to all threads on different NUMA nodes. The only difference for threads on different NUMA nodes is R_{local} and R_{remote}. The thread with the largest R_{remote} will take the longest T_{memory} and the longest total execution time T_{total}. This thread is called the critical thread. We assume $N_{total}*L_{local}$ is equal for all threads. According to the above equations, we have:

$$T_{total} = \alpha * (R_{local} + N_{factor} * R_{remote}) T_{total} + (1-\alpha) T_{total} \tag{8}$$

3.2. Considering the Power Cost of Cores

The power cost of the core is given below:

$$P = P_{dynamic} + P_{leakage} \tag{9}$$

According to [13], the dynamic power cost $E_{dynamic}$ is the cube of the clock frequency of the core and the leakage power cost $E_{leakage}$ is in proportional to the clock frequency of the core. C_d and C_l are the specific parameters, which are only determined by the system hardware. We have the following equations:

$$P_{dynamic} = C_d * f^3 \tag{10}$$

$$P_{leakage} = C_l * f \tag{11}$$

According to equation (9), (10) and (11), we have the following equation:

$$P = C_d * f^3 + C_l * f \tag{12}$$

3.3. Considering the DVFS Adjustment Strategies of Cores

In the following, we consider two different DVFS adjustment Strategies.

3.3.1. Before-hand DVFS Adjustment Strategy

The before-hand DVFS adjustment strategy indicates that the frequency of cores which run non-critical thread is the same to the frequencies of cores which run critical thread. The cores which run non-critical threads will turn off their cores when they have finished their work. In the turn-off stage of cores, the cores only consume the leakage power. Before-hand DVFS adjustment strategy does not need the runtime information, which records the speed of threads during the parallel program execution. The energy cost of the critical thread and non-critical thread is given by the following equations, respectively:

$$E_c = C_d * f_c^3 * T_c + C_l * f_c * T_c \tag{13}$$

$$E_{nc} = C_d * f_{nc}^3 * T_{nc} + C_l * f_{nc} * T_{nc} \tag{14}$$

And,

$$f_c = f_{nc} \tag{15}$$

According to (8), we have:

$$\frac{T_{nc}}{T_c} = \frac{\alpha(R_{l_{nc}} + N_{factor} * R_{r_{nc}}) + (1-\alpha)}{\alpha(R_{l_c} + N_{factor} * R_{r_c}) + (1-\alpha)} \tag{16}$$

According to the equation (8) and (13), (14), (15), we have:

$$E_{nc} = E_c * \frac{\alpha(R_{l_{nc}} + N_{factor} * R_{r_{nc}}) + (1-\alpha)}{\alpha(R_{l_c} + N_{factor} * R_{r_c}) + (1-\alpha)} \tag{17}$$

Assume:

$$\frac{\alpha(R_{l_{nc}} + N_{factor} * R_{r_{nc}}) + (1-\alpha)}{\alpha(R_{l_c} + N_{factor} * R_{r_c}) + (1-\alpha)} = A \tag{18}$$

The saved energy E_{before} of the before-hand DVFS adjustment strategy is:

$$E_{before} = E_c - E_{nc} = (1-A) * E_c \tag{19}$$

If the DVFS adjustment strategy is not used, the local node and remote node both consume E_c energy. If the before-hand DVFS adjustment strategy is used, the local node and remote node consume E_{nc} and E_c energy, respectively. The percent of energy saved is:

$$P_{before} = \frac{2E_c - (E_c + E_{nc})}{2E_c} = \frac{1-A}{2} \tag{20}$$

3.3.2. Back-hand DVFS Adjustment Strategy

The back-hand DVFS adjustment strategy represents that we change the clock frequencies of the cores, which run non-critical threads dynamically, to guarantee the critical thread and non-critical threads can almost arrive at synchronization points simultaneously. This DVFS adjustment strategy needs the runtime information, which records the speed of threads to guide us, to adjust the clock frequencies of the cores that run non-critical threads.

We assume that f_c and f_{nc} are the clock frequencies of the core, which runs the critical thread and non-critical thread respectively. The execution time of the critical thread and non-critical thread is T_c and T_{nc}, respectively. In order to make the execution time of the critical thread and non-critical threads the same, the clock frequency of the core will be proportional to the total execution time of the critical thread and non-critical threads. We have the following equation:

$$\frac{f_{nc}}{f_c} = \frac{T_{nc}}{T_c} = \frac{\alpha(R_{l_{nc}} + N_{factor} * R_{r_{nc}}) + (1-\alpha)}{\alpha(R_{l_c} + N_{factor} * R_{r_c}) + (1-\alpha)} = A \tag{21}$$

We can change the value of f_{nc} to make it satisfy the equation (21), which makes the critical thread and non-critical threads arrive at synchronization points simultaneously:

$$T_c = T_{nc} \tag{22}$$

According to (13), (14), (21) and (22), we have:

$$\begin{aligned} E_{nc} &= C_d * f_{nc}^3 * T_{nc} + C_l * f_{nc} * T_{nc} \\ &= C_d * A^3 * f_c^3 * T_c + C_l * A * f_c * T_c \end{aligned} \tag{23}$$

The saved energy E_{back} of the back-hand DVFS adjustment strategy is:

$$\begin{aligned} E_{back} &= E_c - E_{nc} \\ &= C_d * f_c^3 * T_c * (1 - A^3) + C_l * f_c * T_c (1 - A) \\ &= (1 - A) * (C_d * f_c^3 * T_c + C_l * f_c * T_c) + (A + A^2) * C_d * f_c^3 * T_c \\ &= (1 - A) * E_c + (A + A^2) * C_d * f_c^3 * T_c \end{aligned} \tag{24}$$

Compared with the equation (19) and (24), we can find that the back-end DVFS adjustment strategy can save more energy than the before-hand DVFS adjustment strategy.

In the actual situation, the dynamic energy cost $C_d*f_c^3*T_c$ is much larger than the leakage energy cost $C_l*f_c*T_c$. Therefore, the equation (24) can be approximated to:

$$\begin{aligned} E_{back} &\approx C_d * f_c^3 * T_c * (1 - A^3) + C_l * f_c * T_c (1 - A^3) \\ &= (1 - A^3) * E_c \end{aligned} \tag{25}$$

The percent of energy saved is:

$$P_{back} = \frac{2E_c - (E_c + E_{nc})}{2E_c} = \frac{1 - A^3}{2} \tag{26}$$

4. Simulations

We will compare the energy saved by before-hand and back-hand DVFS adjustment strategy. According to equation (19) and (24), the percent of energy saved is only influenced by A. A is influenced by three factors: α, R_{local} (or R_{remote}) and N_{factor}. In the following, we simulate how the saved energy is influenced by the three factors under before-hand and back-hand DVFS adjustment strategy, respectively.

4.1. Varying the Memory Intensity of Parallel Programs

In this circumstance, the proportion of memory access instructions is the only varying parameter. We assume α is the variable which correspond to the proportion of memory access instructions. R_{l_nc} and R_{r_nc} for the non-critical thread is 75% and 25% respectively. R_{l_c} and R_{r_c} for the critical thread is 25% and 75% respectively. N_{factor} is equal to 1.5 on the basis of the mainstream NUMA systems [23]. According to equation (20) and equation (26), the percent of energy saved, as shown in Figure 4.

As shown in Figure 4, the percent of saved energy increases with the proportion of memory access instructions of the parallel program. The back-hand DVFS adjustment strategy can save more energy the before-hand DVFS adjustment strategy when increasing the proportion of memory access instructions α. If all instructions of the parallel program are memory access instructions (α=1), the before-hand and back-hand DVFS adjustment strategy can save maximally 9.1% and 22.6% of total energy respectively.

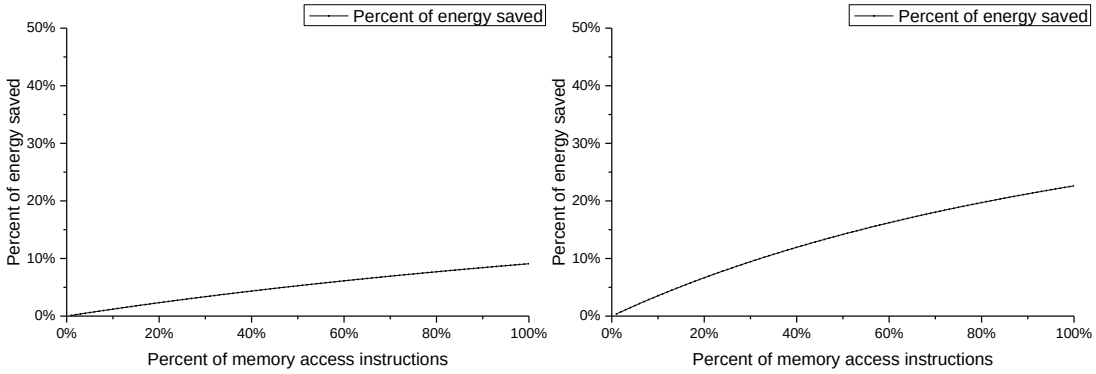


Figure 4. Percent of energy saved when changing the proportion of memory access instructions

4.2. Varying the Proportion of Local and Remote Memory Access

In this case, the proportion of remote memory access is the only varying parameter. R_{r_c} is the variable that corresponds to the proportion of remote memory access of the critical thread. R_{l_c} is equal to 1 minus R_{r_c}. If the data is remote data for critical threads, it will be local data for non-critical threads. We can find out that $R_{r\_c} = R_{l\_nc}$. The proportion of memory access time $\alpha$is assumed to be 50% and N_{factor} is assumed to be 1.5, which is the same as the previous section. According to equation (20) and (26), the percent of saved energy is as shown in Figure 5.

As shown in Figure 5, the percent of saved energy increases with the proportion of remote memory access. The back-hand DVFS adjustment strategy saves more energy than the before-hand DVFS adjustment strategy when increases the proportion of remote memory access R_r. If all memory accesses are remote memory access (R_r =100% and R_l=0%), the before-hand and back-hand DVFS adjustment strategy can save maximally 10% and 24.4% of total energy respectively.

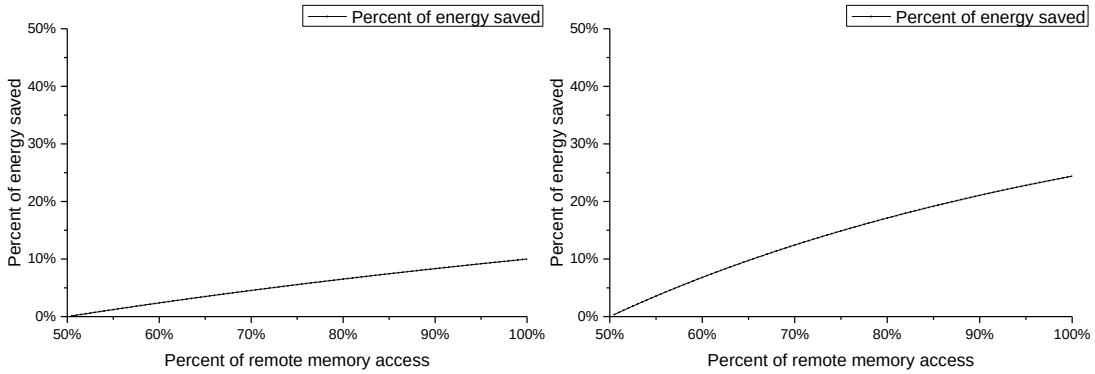


Figure 5. Percent of saved energy when changing the proportion of remote memory access

4.3. Varying the Ratio between Local and Remote Memory Access Latency

In this circumstance, the ratio between remote memory access latency and local memory access latency is the only varying parameter. N_{factor} is the variable which is correspond to ratio between remote and local memory access latency. Based on current mainstream NUMA systems, N_{factor} is between 1.5 to 2 [20]. The scope of N_{factor} is extended from 1 to 5 in our work in order to study how the saved energy changes with N_{factor}. The proportion of memory access instructions is assumed to be 50%. which is the same as the previous section. R_{l_c} and R_{r_c} for critical thread is assumed to be 25% and 75% respectively. R_{l_nc} and R_{r_nc} for non-critical thread is 75% and 25% respectively. According to equation (20) and equation (26), the percent of saved energy is as shown in Figure 6.

As shown in Figure 6, the percent of saved energy increases with the ratio between remote and local memory access latency. The back-hand DVFS adjustment strategy can save more energy the before-hand DVFS adjustment strategy when N_{factor} (the ratio between remote and local memory access latency) is increased. If the latency of the inter-connect link between two NUMA nodes is extremely large (N_{factor}=5), the before-hand and back-hand DVFS adjustment strategy can save maximally 20% and 39.2% of total energy respectively.

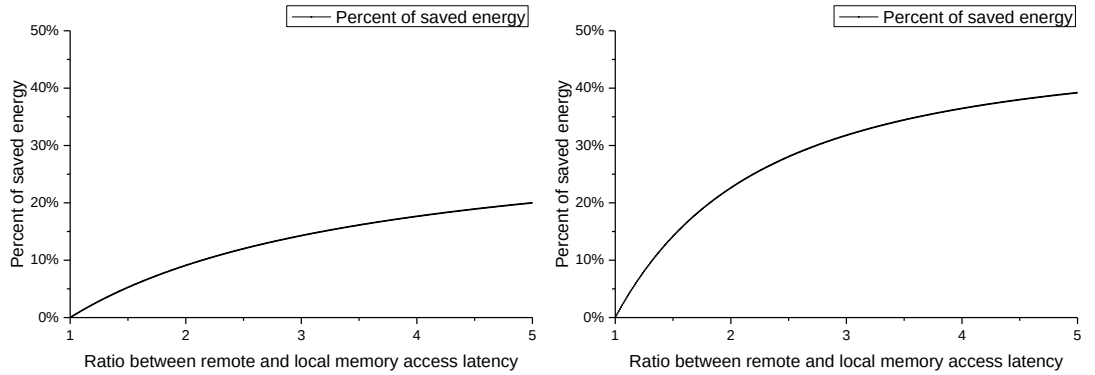
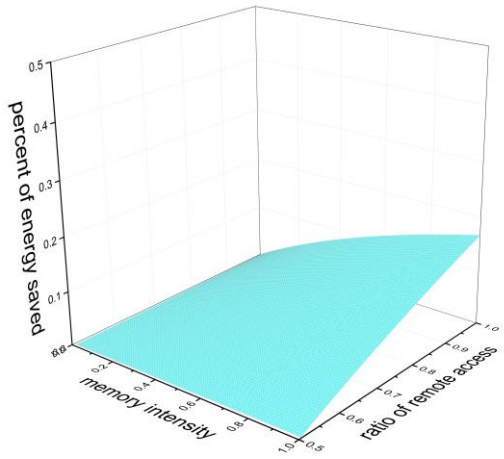
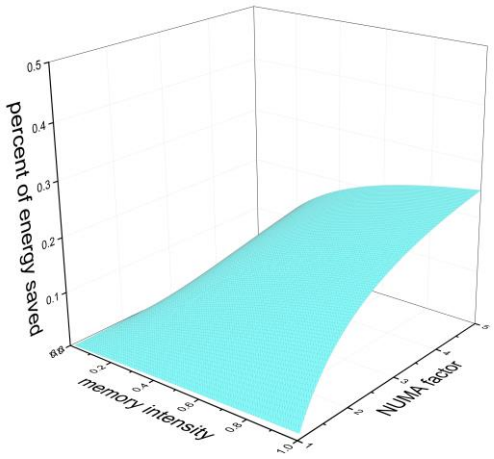


Figure 6. Percent of saved energy when changing the ratio between remote and local memory access latency

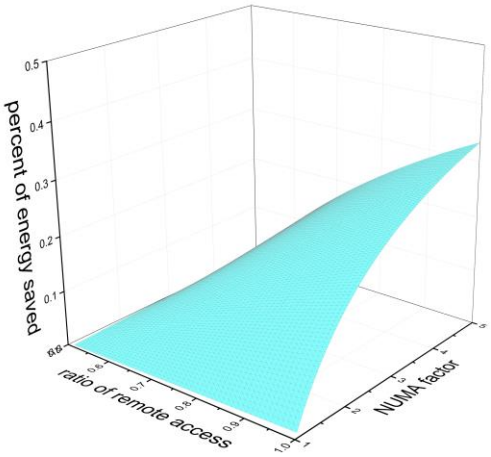
4.4. Considering Three Factors



(a) Changing memory intensity and remote access ratio



(b) Changing memory intensity and N_{factor}



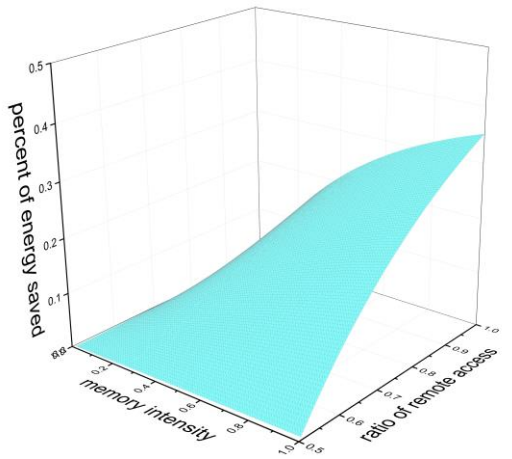
(c) Changing remote access ratio and N_{factor}

Figure 7. Percent of saved energy when changing two factors using before-hand DVFS adjustment strategy

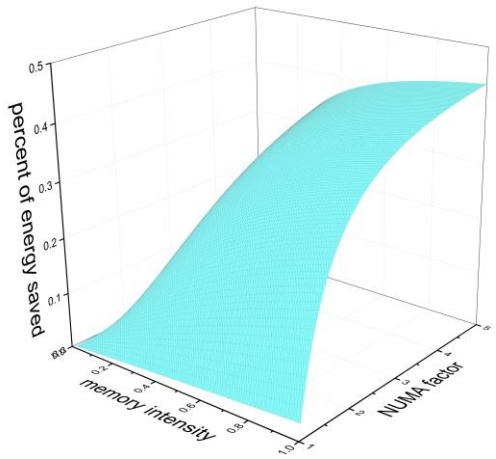
We compare the three factors in the above sections and find that N_{factor} is the most performance influence factor among the three. It means that our DVFS adjustment strategy has the best effect when the latency of inter-connect link between NUMA nodes is very large. These three factors can influence the energy saved by the DVFS adjustment strategy. The ratio between remote and local access latency is the first critical factor which we should consider. If the latency of remote memory access is just a little bigger than the latency of local memory access, the energy saved by DVFS adjustment may be small. Afterwards, we find that α and R_{local} (or R_{remote}) have almost the same influence on the energy saved by DVFS adjustment strategy. From the analysis of Figure 4, 5 and 6, the influence effects of these three factors on the energy saved by DVFS adjustment strategies have the following relations: $N_{factor} > R_{local} \approx \alpha$

What is more, we study how the saved energy changes when two of three factors are variables. In Figure 7(a), 7(b) and 7(c), N_{factor}, R_r and α is assumed to be 1.5, 0.75, 0.5, respectively. The other two factors of the three in each sub-figures are variables. This assumption is the same for Figure 8(a), 8(b) and 8(c).

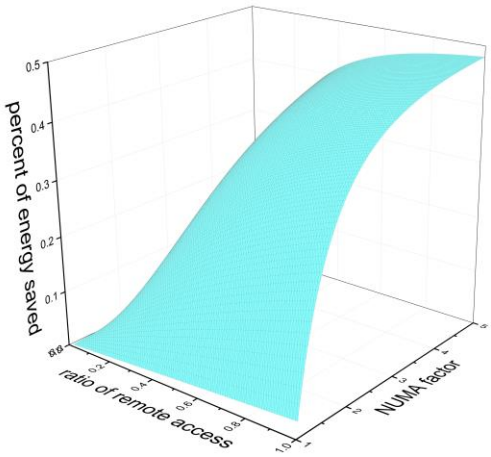
From Figure 7 and 8, we can find out that considering two factors can save more energy than only considering one factors. In Figure 7, we can save maximally 16.7%, 25%, 33.3% of total energy, respectively. In Figure 8, we can save maximally 35.2%, 43.8%, 48.1% of total energy, respectively. The back-hand DVFS adjustment strategy can also save more energy than before-hand DVFS adjustment strategy when we change two factors of three. It can save more energy when these two factors include N_{factor} which is the ratio between remote and local memory access latency.



(a) Changing memory intensity and remote access ratio



(b) Changing memory intensity and N_{factor}



(c) Changing remote access ratio and N_{factor}

Figure 8. Percent of saved energy when changing two factors using back-hand DVFS adjustment strategy

5. Relate studies

Previous research has focused on optimizing the performance of NUMA multi-core systems. A rich collection of research work focused on addressing the performance bottlenecks caused by shared resource contention. The cores on the same NUMA node compete for the on-chip shared memory resources (such as memory controllers, instruction pre-fetch units and last-level cache), which is discussed in [7,9,10,28]. Moreover, a large amount of research work extensively studied the performance problem caused by data locality on NUMA multi-core systems. The data locality problem is important on NUMA multi-core systems because that remote data access has to go across the inter-connect links and remote memory controller. It takes more time and has lower data-transfer bandwidth than local memory access, as discussed in [18,19,24]. However, there are rare research work which discusses about reducing energy cost of multi-threaded programs on NUMA systems.

One of the most similar research work with ours was taken by Su et al. [23]. They proposed an optimal thread mapping algorithm which considered the local memory access and critical path problem. The critical path was calculated by the number of local memory access and remote memory access. The algorithm used the critical path information to guide thread mapping. Thread was mapped onto the memory node which had the lowest critical path. They used the thread mapping mechanism to solve the workload imbalance caused by the NUMA characteristics. However, we use the DVFS adjustment strategy to solve the workload imbalance caused by the NUMA characteristics. Our method is more suitable for the parallel algorithms which use static mapping mechanism.

Another similar research work with ours was proposed by Cai et al. [8]. They proposed a novel mechanism, which used marks to label how many loops had been executed in an OpenMP parallel region. The thread with the least number of executed parallel loops was called the critical thread, which lagged down the overall performance of the multi-threaded program. They also used the DVFS adjustment strategy to speed up the critical thread and slow down non-critical threads. The experiments showed that their mechanism could save a considerable amount of energy while causes negligible

performance loss. Their mechanism collected the runtime information which recorded the number of executed parallel loops to predict the critical degrees of threads on multi-core systems. Nevertheless, our method considers about three factors which could influence the critical degrees of threads on NUMA architectures. Their mechanism needed to modify the source code of the multi-threaded program. Our strategy only needed to record the runtime information of the multi-threaded program and did not need to modify the source code.

Bhattacharjee et al. [4] proposed a predictor which was used to predict critical threads of a multi-threaded program. Their predictor only used the runtime information about L1 instruction, L1 data and L2 cache misses to predict the critical degree of threads. Their predictor had an average accuracy of 93% across a wide range of architectures and benchmarks. In contrast, our model was used to predict the critical degree of threads by using the information about the memory intensity of the parallel program, the ratio of remote access and ratio between remote and local access latency. Our model could be more accurate on NUMA architectures due to the fact that we considered the factors that were related with NUMA characteristics.

6. Conclusions

Rare research work has considered about reducing the energy cost of multi-threaded programs on NUMA architectures. Our work considers energy cost caused by the critical thread of multi-threaded programs on NUMA architectures. Three factors can influence the critical degree of threads: (1) the memory intensity of parallel programs; (2) the proportion of remote memory accesses; (3) the ratio between remote and local memory access latency. Afterwards, the DVFS adjustment strategy can be used to slow down the speed of non-critical threads so as to make the critical thread and non-critical threads arrive at the synchronization points simultaneously. In this way, we can save the energy because of the lower frequency of the CPU cores which run non-critical threads. Before-hand DVFS adjustment strategy can save maximally 20%, 33.3% of total energy and Back-hand DVFS adjustment strategy can save maximally 39.2% and 48.1% of total energy, when considering one factor and two factors respectively.

Acknowledgments

This study is supported in part by the Natural Science Foundation of Hubei Province of China under Grant No.2017CFC889, and the youth foundation of Wuhan Donghu University under Grant No.2017dhzk007.

References

1. A. E. Abdallah, E. E. Abdallah, F. Hanandeh, A. Aljammal and E. Al-Daoud, "Power Aware Ant Colony Routing Algorithm for Mobile Ad-hoc Networks," *International Journal of Software Engineering and Its Applications*, vol.9, no.12, pp.197-212, 2015
2. S. Albers, "Energy-Efficient Algorithms," *Communications of the ACM*, vol.53, no.5, pp.86-96, 2010
3. E. Ayguade, N. Copty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang, "The Design of OpenMP Tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol.20, no.3, pp.404-418, 2009
4. A. Bhattacharjee and M. Martonosi, "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors," *In ACM SIGARCH Computer Architecture News*, vol.37, pp.290-301, 2009
5. C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The Parsec Benchmark Suite: Characterization and Architectural Implications," *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, New York, NY, USA, pp.72-81, 2008
6. H. R. Boveiri, "An Efficient Task Priority Measurement for List-Scheduling in Multiprocessor Environments," *International Journal of Software Engineering and Its Applications*, vol.9, no.5, pp.233-246, 2015
7. S. Blagodurov, S. Zhuravlev, A. Fedorova, and A. Kamali, "A Case for NUMA-Aware Contention Management on Multicore Systems," *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, New York, NY, USA, pp.557-558, 2010
8. Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González, "Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions," *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, New York, NY, USA, pp.240-249, 2008
9. M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, "Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems," *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, New York, NY, USA, pp.381-394, 2013
10. T. Dey, W. Wang, J. Davidson, and M. Soffa, "Characterizing Multi-threaded Applications Based on Shared-Resource Contention," *Proceedings of the 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp.76-86, 2011
11. D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems," *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, New York, NY, USA, pp.413-422, 2009
12. Y. S. Hwang and J. M. Jeong, "Design of a Multi-Threaded Image Signal Processor with a Multi-Bank Cache Memory,"

International Journal of Software Engineering and Its Applications, vol.10, no.9, pp.1-8, 2016
13. V. A. Korthikanti and G. Agha, "Energy-Performance Trade-off Analysis of Parallel Algorithms," *USENIX Workshop on Hot Topics in Parallelism*, 2010
14. N. B. Lakshminarayana, J. Lee, and H. Kim, "Age Based Scheduling for Asymmetric Multiprocessors," *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, New York, NY, USA, pp.25:1-25:12, 2009
15. Lin Y, Zhu X, Zheng Z, et al. "*The individual identification method of wireless device based on dimensionality reduction and machine learning*". Journal of Supercomputing, No.5, pp.1-18(2017).
16. Lin Y, Wang C, Ma C, et al: "*A new combination method for multisensor conflict information*," Journal of Supercomputing, Vol.72, No.7, pp. 2874-2890 (2016)
17. Yun Lin, Chao Wang, Jiaxing Wang, Zheng Dou. "*A Novel Dynamic Spectrum Access Framework Based on Reinforcement Learning for Cognitive Radio Sensor Networks*". Sensors, Vol.16, No.10, pp. 1-22(2016).
18. Q. Lu, C. Alias, U. Bondhugula, T. Henretty, S. Krishnamoorthy, J. Ramanujam, A. Rountev, P. Sadayappan, Y. Chen, H. Lin, and T.-F. Ngai, "Data Layout Transformation for Enhancing Data Locality on NUCA Chip Multiprocessors," *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp.348-357, 2009
19. Z. Majo and T. R. Gross, "Memory Management in NUMA Multicore Systems: Trapped Between Cache Contention and Interconnect Overhead," *Proceedings of the 2011 International Symposium on Memory Management (ISMM)*, New York, NY, USA, pp.11-20, 2011
20. C. McCurdy and J. Vetter, "Memphis: Finding and Fixing NUMA-Related Performance Problems on Multi-core Platforms," *Proceedings of the International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pp.87-96, 2010
21. M. N. A. Rahman, A. F. A. Nasir, N. Mat and A. R. Mamat, "Image Segmentation using OpenMP and its Application in Plant Species Classification," *International Journal of Software Engineering and Its Applications*, vol.9, no.5, pp.135-144, 2015
22. Shi C, Dou Z, Lin Y, et al. "*Dynamic threshold-setting for RF-powered cognitive radio networks in non-Gaussian noise*". Physical Communication, Vol. 27, No. 4, pp. 99-105, 2018
23. C. Su, D. Li, D. S. Nikolopoulos, M. Grove, K. Cameron, and B. R. De Supinski, "Critical Path-Based Thread Placement for NUMA Systems," *ACM SIGMETRICS Performance Evaluation Review*, vol.40, no.2, pp.106-112, 2012
24. D. Tam, R. Azimi, and M. Stumm, "Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors," *ACM SIGOPS Operating Systems Review*, vol.41, no.3, pp.47-58, 2007
25. V. Thind, B. Pandey, K. Kalia, T. Das and T. Kumar, "FPGA Based Low Power DES Algorithm Design and Implementation using HTML Technology," *International Journal of Software Engineering and Its Applications*, vol.10, no.6, pp.81-92, 2016
26. Wu Q, Li Y, Lin Y: "*The application of nonlocal total variation in image denoising for mobile transmission*," Multimedia Tools & Applications, Vol.76, No.16, pp. 1-13 (2016)
27. J. Yao, "Numatop: A Tool for Memory Access Locality Characterization and Analysis," *Intel Open Source Technology Center*, 2013
28. S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," *ACM SIGARCH Computer Architecture News*, vol.38, no.1, pp.129-142, 2010