2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing

Energy-Aware Loop Parallelism Maximization for Multi-Core DSP Architectures

Meikang Qiu[1] Jian-Wei Niu[2] Laurence T. Yang[3] Xiao Qin [4] Senlin Zhang[5] Bin Wang[6]
[1]Department of Electrical and Computer Engineering, University of Kentucky, Lexington, KY 40506, mqiu@engr.uky.edu
[2]State Key Lab of Software Develop. Environment, Beihang University, Beijing 100191, China, niujianwei@buaa.edu.cn
[3]Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada, ltyang@stfx.ca
[4]Dept. of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA, xqin@auburn.edu
[5]Dept. of Systems Science and Engineering, Zhejiang University, Hangzhou 310027, China, slzhang@zju.edu.cn
[6]Dept. of Computer Science and Engineering, Wright State University, Dayton, OH 45435, USA, bin.wang@wright.edu

***Abstract*— With the advance of semiconductor, multi-core architecture is inevitable in today's embedded system design. Nested loops are usually the most critical part in multimedia and high performance DSP (*Digital Signal Processing*) systems. Hence, maximizing loop parallelism is an important issue to improve the performance of a modern compiler. This paper studies how to maximize the system performance with the consideration of energy reduction for applications with multidimensional nested loops on multi-core DSP architectures. An algorithm, EALPM (*Energy-Aware Loop Parallelism Maximization*), is proposed in this paper. We implemented a two phase strategy. First, the strategy uses retiming and loop transformation to parallelize nested loops. Then the strategy employs a novel voltage assignment algorithm to reduce total energy consumption. The experimental results show that on average, both performance and energy-saving can be significantly improved using the EALPM algorithm.**

***Index Terms*— Energy, multi-core, loop parallelization, retiming, voltage assignment**

I. Introduction

With the advent of parallel architectures and systems with deep memory hierarchies, nested loops optimization has become an important area in high-level optimizations. In many DSP applications such as multidimensional signal processing and video applications, nested loops are the most critical sections in which most of the execution time and power is spent. The harness of parallelism in programs to fully utilize their computation power is so important that this can never be over-emphasized. Although we can use multiple-threaded tools to generate coarse-grain parallel programs [1], [2], it is hard to implement loop parallelization at the granularities of loop iterations level manually. Many automatic loop parallelization techniques have been developed in the previous work [3], [4].

This paper focuses on improving both performance and energy saving for loop applications. We choose to target loop optimizations for two reasons [5]: First, through loop optimizations, the multimedia and signal processing applications operated on multidimensional array structures will improve their performance and energy saving. Second, these optimizations are widely used by commercial and academic optimizing compilers. For example, TI TMS320DM6467 (with Dual-Core) is widely used in DSP applications. To the best of our knowledge, this paper is the first one that addresses improving performance through maximizing loop parallelism and reducing energy consumption through voltage island assignments for nested loops on multi-core architecture.

There are two kinds of parallelism: 1). Data parallelism, involves performing a similar computation on many data objects simultaneously [6]. 2). Task parallelism, involves performing different tasks in parallel, where a task is an arbitrary sequence of computation. Task parallelism, a challenging issue, is one of the two major focuses of this paper. For loops, dependence cycles are the main obstacle to parallelization. The dependence relation in a set of instructions constructs a dependence cycle. In this paper, we will combine the iteration level loop parallelization technique (see [2]) with voltage levels assignment to increase both performance and energy-saving for multi-core architecture. We target at iteration-level parallelism, which means different iterations from the same loop kernel can be executed in parallel. In this approach, loop transformation is modeled by retiming [7], [8]. Retiming was originally proposed to minimize the cycle period of a synchronous circuit by evenly distributing registers. Sha et al. has extended it to schedule data flow graphs on parallel systems [9].

In [2], an iteration level algorithm has been proposed. The basic idea is to migrate inter-iteration data dependencies by regrouping statements of a loop kernel in such a way that the number of consecutive independent iterations is always maximized. This approach constructs a dependence graph to model the data dependencies among the statements in a loop and then uses retiming to model dependence migration among the edges in the dependence graph. As a result, this classic loop optimization problem is transformed into a graph optimization problem, i.e., one of finding retiming values for its nodes so that the minimum non-zero edge weight in the graph is maximized. The scheme proposed by Shao et al. [2] does not consider energy consumption while achieving the loop parallelism.

DVS (Dynamic Voltage Scaling) allows CPU frequency and voltage to change dynamically at run time, in order to match demand. The Transmeta Crusoe TM5800 processor can scale its frequency from 800MHz down to 367MHz and its voltage from 1.3V down to 0.9V, thereby reducing power consumption [10]. DVS is also employed on Intel XScale processors [11]. Because required voltage scales roughly linearly with frequency within typical operating ranges, and energy is proportional to the square of the voltage, a modest reduction in supply voltage can yield significant energy-saving [12], [13]. DVS allows the processor to slow down to match the speed of the real world or external bottleneck. Recently, researchers have proposed globally asynchronous and locally synchronous DVS processors in which the frequency and voltage of various processor components ("domains") can be changed independently at run time [14]–[17].

Combining the consideration of energy and performance, in the study, we design an algorithm, EALPM (*Energy-Aware Loop Parallelism Maximization*), to improve both performance and energy saving. The experimental results show that EALPM significantly improves both performance and energy saving. For example, with 8 cores, EALPM demonstrates an average 86.5% reduction of total execution time, compared with the approach without using parallelism (Med-P1). Compared with the approach without using voltage assignment (Med.3), EALPM shows an average 28.7% reduction in total energy consumption.

In summary, our paper has three major contributions: First, we use retiming and loop transformation to parallelize nested loops. Second, we use a novel voltage assignment algorithm to reduce total energy consumption. We have considered to reduce leakage power by system level power management. Third, data parallelism is employed to increase the performance.

In the next section, we will introduce necessary background with examples. Then, we discuss the algorithms in Section III. We demonstrate our experimental results in Section IV. Concluding remarks are provided in Section V.

II. Basic Concepts and Models

In this section, we introduce some basic concepts, which will be used in the later sections. First, we introduce parallelism models coupled with an example. Next, we introduce the MLDG (*Multi-Dimensional Loop Dependency Graph*) model and multidimensional retiming. Finally, we describe the energy model.

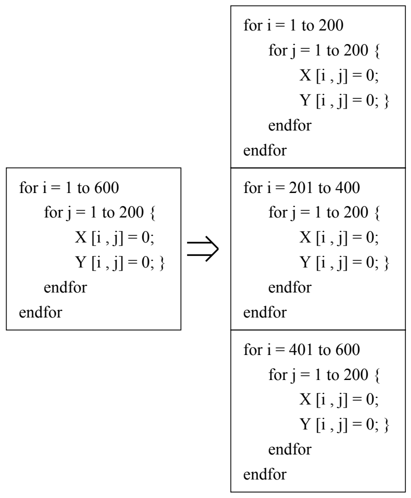
A. Parallelism Models

Embedded systems usually have strict timing and memory constraints. Effective techniques targeting at embedded systems must consider both factors. Loops are the most time-consuming parts of the computation-intensive applications for embedded systems, so it is important to optimize the execution of loops [12].

There are two kinds of parallelism. The first one is called data parallelism, which involves performing a similar computation on many data objects simultaneously. For nested loops, this corresponds to several cores executing a given nested loop in parallel. All the processors execute a similar program (the same loop body), but work on different parts of array data, i.e., they execute different iterations of the loop [6]. In order to map a given multi-media program to a multi-core architecture, we address both data parallelism and task parallelism issues.

Figure 1 shows an example of data parallelism. The program on the left-hand side is a two-dimensional nested loop. For the outside loop i, the iteration number is 600. We can divide the 600 into three 200 iterations, i.e., 1 to 200, 201 to 400, and 401 to 600. Each part will be implemented by a separate core. The three parts do not have dependencies, hence they can be executed in parallel. This example illustrates the general idea of data parallelism.

The second type of parallelism is task parallelism, which involves performing different tasks in parallel, where a task is an arbitrary sequence of computation. For nested loops, this type of parallelism represents executing different loop nests in different cores at the same time [6]. For example, in Figure 1, instructions $X[i, j] = 0$; and $Y[i, j] = 0$; appear



```
for i = 1 to 600
    for j = 1 to 200 {
        X [i , j] = 0;
        Y [i , j] = 0; }
    endfor
endfor
```

⇒

```
for i = 1 to 200
    for j = 1 to 200 {
        X [i , j] = 0;
        Y [i , j] = 0; }
    endfor
endfor
```

```
for i = 201 to 400
    for j = 1 to 200 {
        X [i , j] = 0;
        Y [i , j] = 0; }
    endfor
endfor
```

```
for i = 401 to 600
    for j = 1 to 200 {
        X [i , j] = 0;
        Y [i , j] = 0; }
    endfor
endfor
```

Fig. 1. An example of data parallelism.

in the same loop body. If we want to execute them on different cores to improve performance, iteration level parallelism must be achieved. This is a more difficult problem compared with the parallelization of outside loop i. In the next two subsections, we will introduce the background of *Multi-Dimensional Loop Dependency Graph* (MLDG) and retiming issues. We will show how to solve task parallelism for nested loop with a detailed example.

B. Multi-Dimensional Loop Dependency Graph (MLDG)

To clearly show the data dependencies between loops, we use a *multi-dimensional loop dependency graph* (MLDG) to model a nested loop. A MLDG G = (V, E_d, δ) is a node-weighted and edge-weighted directed graph, where V is a set of nodes representing the loops to be fused. $E_d \subseteq V \times V$ is a set of edges representing dependencies between the loops. δ is a function from E_d to Z^n representing the minimum loop dependency vector between two loops [2], [12], [18].

We use the minimum loop dependency vector of an edge in the MLDG model instead of all the loop dependency vectors to improve the efficiency of our technique. All the comparisons between two loop dependency vectors are based on the lexicographic order in this paper. For example, in the two-dimensional case, a vector $\vec{v} = (v_1, v_2)$ is smaller than a vector $\vec{u} = (u_1, u_2)$ according to the lexicographic order if either $v_1 < u_1$ or $v_1 = u_1$ and $v_2 < u_2$ [12], [19].

C. Multi-Dimensional Retiming

A graph transformation technique is used in this study to remove parallelism-prevention dependencies based on the multidimensional retiming technique. Note that the retiming technique preserves all the data dependencies of the original *Loop Dependency Graph* (LDG). Retiming redistributes delays, i.e., data dependency distances, in a graph to achieve parallelism. Retiming and Dependence Migration Retiming [7] is used to model dependence migration, and it is defined as follows.

Definition 2.1: Given a dependence graph $G = (V, E, w)$, a retiming r of G is a function that maps each node in V to an integer $r(v)$. For a node $u \in V$, the retiming value $r(u)$ is the number of dependence distances (edge weights) drawn from each of its incoming edges and pushed to each of its outgoing edges. Given a retiming function r, let $G_r = \langle V, E, w_r \rangle$ be the retimed graph of G obtained by applying r to G. Then $w_r(u, v) = w(u, v) + r(u) - r(v)$ for each edge $(u, v) \in E$ in G_r [12].

For a multi-dimensional loop dependency graph G, a multidimensional retiming $\vec{r}$ is a function from V to Z^n. The retiming value $\vec{r}(u)$ represents how many delays are added into the edges $u \to v$ and subtracted from the edges $w \to u$, for $u, v, w \in V$. Therefore, in the retimed MLDG G^r, we have $\delta^r(e) = \delta(e) + \vec{r}(u) - \vec{r}(v)$ for each edge $e : u \to v$. The summation of the edge weights in a cycle remains a constant after retiming. The retiming value $\vec{r}(u)$ means that a node u originally executed in the iteration $\vec{i}$ is moved to the iteration $\vec{i} - \vec{r}(u)$. For a loop dependency graph, all the computation of loop u are executed $\vec{r}(u)$ iterations earlier. Some iterations of the original loop are moved out of the loop body to become prologue and epilogue. More specifically, the codes to be executed before and after the loop body to complete the execution of the whole loop. The number of copies of a node u in prologue or epilogue can be computed from the retiming value [12].

The normalized retiming value for node u is defined as $r(u) - min_u r(u)$, where $min_u r(u)$ is the minimum retiming value of all nodes u in V. From this definition, we know that the normalized retiming value for any node in V is larger than or equal to (0, 0) in the two-dimensional case.

Figure 2 shows an example of iteration level parallelism. In Figure 2(a), there is a loop i with two instructions $M1$ and $M2$. Figure 2(b) shows the loop dependency graph of these two instructions. For $M1$, $A[i]$ waits for one-iteration-before information of B, i.e., $B[i - 1]$. Thus, we have an

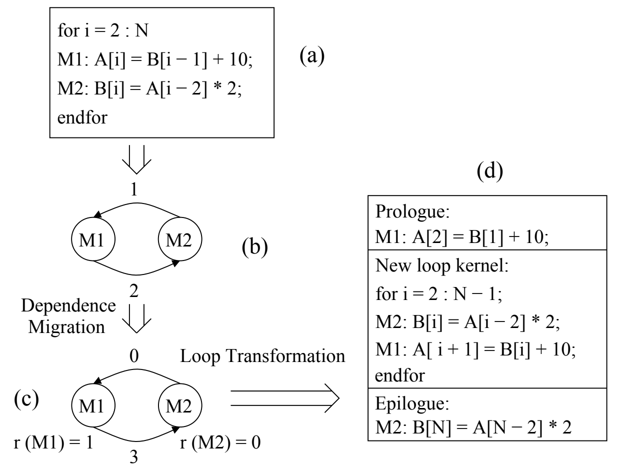


Fig. 2. An example of iteration level loop parallelism.

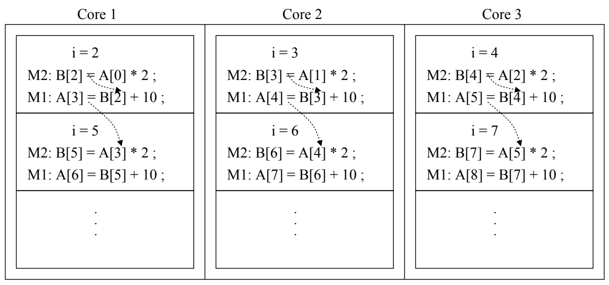


Fig. 3. An example of iteration level loop parallelism.

arrow with 1 going to $M1$. Similarly, we have an arrow with 2 going to $M2$, which means $M2$ needs to wait for two-iteration-before information from A. Figure 2(c) shows how to use retiming to change the dependencies of the two instructions. With retiming $r(M1) = 1$, for $M1$, the incoming number 1 is shifted to the outgoing arc, then the number of outgoing arcs is increased to 3. Then the value of incoming arc of $M1$ becomes 0. For $M2$, we need to use retiming, so we have $r(M2) = 0$. Figure 2(d) shows the actually changes in the program shown in Figure 2(a) after implementing the retiming technique. The whole loop into is divided into three parts. The prologue part only has one instruction, $M1 : A[2] = B[1] + 10$;. The epilogue part also only has one instruction, $M2 : B[N] = A[N-2] * 2$;. The major part, i.e., the new kernel part, is in the middle. We can use three cores to implement this major part in parallel. The execution time is reduced to nearly 1/3 of the original approach.

Figure 3 shows how to implement the new loop kernel part with three cores in parallel: core 1 executes iterations 2, 5, 8, 11, $\cdots$; core 2 performs iterations 3, 6, 9, 12, $\cdots$; and core 3 implement iterations 4, 7, 10, 13, $\cdots$. In iteration 2, A[3] is generated, and it is needed only in iteration 5. $B[2]$ can be directly used in the same iteration 2. Hence, after retiming, there is no dependency problem to prevent the parallelization. The non-zero number 3 actually decides how many cores can be used in parallel. Paper [2] has detailed proofs and shows how to retime the graph and implement the program on arbitrary number of cores.

D. The Energy Model

Energy consumption needs to be considered as one of the primary metrics in embedded system design. In a CMOS circuit, The dynamic power consumption is proportional to $V^2 \cdot f \cdot C_L$, where C_L is the load capacitance, V is the supply voltage, and f is the system clock frequency [20]–[25]. Therefore, energy is equal to $V^2 \cdot f \cdot c \cdot t$, where c is a constant and different components may have different c. Reducing the supply voltage can result in substantial power and energy saving. Roughly speaking, system's power dissipation will be halved if we reduce V by 30% without changing any other system parameters [12], [21].

According to the α formula in CMOS circuit, the cycle period time T_c is proportional to $\frac{V}{(V-V_{th})^{\alpha}}$, where V_{th} is the threshold voltage and $\alpha \in$ (1.0,2.0] is a technology dependent constant [21]–[24]. Given the number of cycles N of node v, its computation time $T(v) = N \times T_c$. We can see that the lower voltage will prolong the execution time of a node but reduce its energy consumption.

DVS (Dynamic Voltage Scaling) is a technique that varies system's operating voltages and clock frequencies based on the computation load to provide desired performance with the minimum energy consumption. It has been demonstrated as one of the most effective low power system design techniques and has been supported by many modern microprocessors. Examples include Transmeta's Crusoe, AMD's K-6, Intel's XScale and Pentium III and IV, and some DSPs developed in Bell Labs [12].

One hardware approach to save energy is the voltage islands technique [6], [26], [27]. The core-based design using voltage islands is a new technique which helps reducing both switching and standby components of power dissipation. Simply speaking, a voltage island is a group of on-chip cores powered by the same voltage source, independently from the chip-level voltage supply. The use of voltage islands permits operating different portions of the design at different voltage levels in order to optimize the overall chip power consumption. In the SoC (System on Chip) context, the voltage island enables core-level power optimization by utilizing a power supply that is unique from the rest of the design.

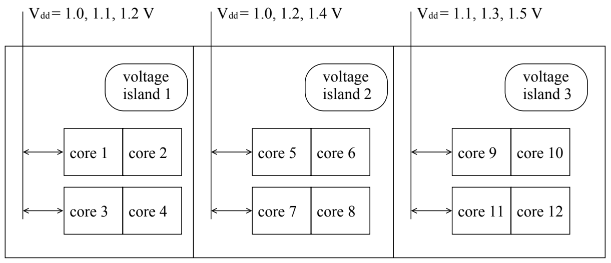


Fig. 4. The multi-core architecture with voltage islands.

Figure 4 shows the multi-core architecture with voltage islands. In this architecture, the chip area is divided into multiple voltage islands, each of which is controlled by a separate power feed and operates under a different voltage level/frequency. There are three voltage islands shown in Figure 4. The V_{dd} in island 1 has three voltage levels: 1.0 V, 1.1 V, and 1.2 V; island 2 has 1.0 V, 1.2 V, and 1.4 V and island 3 has 1.1 V, 1.3 V, and 1.5 V; Each voltage island is divided into multiple power domains. All the domains within an island are fed by same V_{dd} source but independently controlled. For example, in Figure 4, island 1 has two domains, each domain has two cores. So totally 4 cores, i.e., cores 1, 2, 3, and 4, are in island 1. The advantage of this island based architecture is that it can help save both dynamic and leakage power [27]. Specifically, it can save dynamic energy by employing different voltage levels (using DVS) for the different islands, and reduce leakage energy by shutting down the power domains that are not needed by the current computation, and other methods such as using ABB (Adaptive Body Biasing) [28].

We define the voltage assignment problem as follows: given N core on a heterogeneous multi-core architecture, i.e., the number of power domain of each voltage island need not be the same; the number of cores in each power domain also is not necessary to be the same; and the voltage levels of each V_{dd} power line are heterogeneous, how to select the cores, power domain, and voltage islands and assign the proper voltage level to each core to achieve the best performance and energy saving.

III. The Algorithms

In this section, an algorithm, EALPM (*Energy-Aware Loop Parallelism Maximization*), is designed to improve the performance and energy saving by loop parallelism maximization and voltage assignments for nested loops.

A. The EALPM Algorithm

Algorithm III.1 EALPM

```
Require: A heterogeneous multi-core platform with $N$ cores, $M$ independent tasks.
Ensure: Voltage assignment $A$ and the retiming $\vec{r}$, to MIN($E$).
 1: Divide the outside loop by using data parallelism so that multiple cores execute different segments of a given loop in parallel.
 2: Use dependency migration algorithm [2] to find a retiming function for a given dependence graph so that the minimum non-zero edge weight in the retimed graph is maximized.
 3: Implement task parallelism, based on the number of parts without dependencies among them.
 4: Record the retime $\vec{r}$.
 5: Based on the obtained new loop structure, implement our voltage assignment algorithm $VA$. Assign proper voltage to each core so that the total energy is minimized.
 6: Shut down the unused voltage islands and power domains to reduce leakage energy.
 7: Use ABB method to reduce leakage energy for active cores.
 8: Record the new assignment $A$.
 9: Record $E_{min}$.
10: Output $A$, $\vec{r}$, and $E_{min}$.
```

The EALPM algorithm is shown in Algorithm III.1. We first implement data parallelism for outside loop. Next, use dependency migration algorithm [2] to find the maximum loop parallelism at iteration level. Based on the obtained loop structure, we do task parallelism. Based on the obtained new loop structure, we implement our voltage assignment algorithm, which is shown in Algorithm III.2. Then we turn off the unused voltage islands and power domains in any voltage island to reduce leakage energy. We also use ABB to reduce leakage energy for active cores. Finally, output the assignment A, retime r, and the minimal energy consumption E_{min}.

The EALPM algorithm has several advantages: 1). Use dependency migration algorithm to maximize iteration parallelism for a loop. 2). Exploit the task and data parallelism at system level. 3). Use voltage assignment to save dynamic energy and power management to reduce leakage energy while achieve maximum performance.

B. The VA Algorithm

Algorithm III.2 VA

```
Require: A heterogeneous multi-core platform with $N$ cores, $M$ independent tasks.
Ensure: Voltage assignment $A$.
 1: Check core availability, how many cores are in active mode and not being used, noted as $N_1$.
 2: if $N_1 < M$ then
 3:     Activate cores until the total active cores are $M$.
 4: end if
 5: if $N < M$ then
 6:     Merge some tasks and activate all cores until $N \geq M$.
 7: else
 8:     Predict the workload $W_i$, where $1 \leq i \leq N$, for each task.
 9:     Sort the tasks according to their workload in descending order.
10:     Assign the task with highest load to the core with highest possible voltage available.
11:     Assign other tasks in the same power domain that has cores already been assigned tasks, until the power domain is full.
12:     Assign other tasks in the same voltage island until the voltage island is full.
13:     Predict the execution time or finish time $T$ for the core with the highest task load using the highest voltage level.
14:     Select the proper voltage level for each core so that the execution time of each core $t_i \leq T$, where $2 \leq i \leq M$.
15:     Output the voltage assignment $A$, which consists of voltage level of each core.
16: end if
```

When it comes to voltage assignment on a heterogeneous multi-core platform, we develop algorithm *VA*, which is shown in Algorithm III.2. First, we check how many cores are in active mode. Then we arrange the number of active cores to be the same as the number of tasks. If the total number of cores is less than the number of tasks, we need to merge some tasks. Next, predict the workload on each core. Assign the task with the highest load to the core with the highest possible voltage available and put other tasks on the cores in the same domain and the same island. Then we predict the execution time of the first selected core and assign proper voltage for other cores so that these cores will not finish early in order to reduce total energy consumption. Finally, output the voltage assignment A.

IV. Experiments

We conduct experiments with the EALPM algorithm on a set of benchmarks including All-pole filter, Differential Pulse-Code Modulation device (DPCM), Wave Digital filter (WDF), Infinite Impulse filter (IIR), Floyd-Steinberg algorithm (Floyd), and Two dimensional filter (2D). We build a simulation framework to evaluate the effectiveness of our approach. The experiments are performed on a Dell Optiplex with a Intel Core 2 Quad 2.83 GHz processor and 3 GB memory running Red Hat Linux 9.0.

We first conduct experiments focusing on six methods to compare the total energy consumption. Method 1: Data parallelism; Method 2: Task parallelism Method 3: Both data parallelism and task parallelism; Method 4: Data parallelism with voltage assignment; Method 5: Task parallelism with voltage assignment; Method 6: Our EALPM algorithm.

The experimental results for the six methods are shown in Table I when the number of cores is 8. Entries "Med.1" to "Med.6" represent the six methods we used in the experiments. Column "E" represents the minimum total energy consumption obtained from six different methods. Column "% M1" to "% M5" represents the percentage of reduction in total energy consumption, compared to the Method 6 (our algorithm), respectively. The average reduction is shown in the last row of the table.

The results show our EALPM can significantly improve energy efficiency for multi-core architecture. Our algorithms improve the energy reduction over other five methods. For example, compared with the approach without using voltage assignment (Med.3), EALPM shows an average 28.7% reduction in total energy consumption. With other experiments, we observe that with more cores available, the reduction ratio for the total energy consumption has increased.

We compare the performance with four other methods. Method 1: Program implementation without task parallelism and data parallelism. Method 2: Data parallelism only. Method 3: Task parallelism only. Method 4: Our EALPM

Six Methods Energy Comparison with 8 cores												
Bench.	N.	Med.1	% M1	Med.2	% M2	Med.3	% M3	Med.4	% M4	Med.5	% M5	Med.6
		E(mJ)		E(mJ)		E(mJ)		E(mJ)		E(mJ)		E(mJ)
IIR	160	426	38.5	448	41.5	361	27.4	329	20.4	349	24.9	262
DPCM	160	442	37.8	472	41.7	392	29.7	341	19.4	376	26.9	275
Floyd	160	452	39.6	468	41.7	381	28.3	347	21.3	355	23.1	273
All-pole	290	727	39.2	759	41.8	620	28.7	559	20.9	568	22.2	442
WDF(1)	40	1088	37.6	1145	40.7	952	28.7	853	20.4	881	22.9	679
WDF(2)	120	326	37.7	335	39.4	289	29.8	256	20.7	265	23.4	203
2D(1)	340	919	39.0	945	40.6	788	28.8	707	20.7	721	22.2	561
2D(2)	40	1251	38.0	1357	42.9	1076	28.0	957	19.0	1012	23.4	775
MDFG1	80	239	38.1	278	46.8	201	26.4	181	18.2	190	22.1	148
MDFG2	80	276	38.4	278	38.8	234	27.4	218	22.0	229	25.8	170
Average Reduction (%)			38.4		41.6		28.7		20.3		23.7	

TABLE I

THE COMPARISON OF TOTAL ENERGY CONSUMPTION WITH SIX METHODS ON VARIOUS BENCHMARKS FOR 8 CORES.

Four Methods performance Comparison with 8 cores								
Bench.	N.	M-1	M-2	% M1	M-3	% M1	M-4	% M1
		T(mS)	T(mS)		T(mS)		T(mS)	
IIR	160	190	58	69.5	79	58.4	32	83.2
DPCM	160	179	46	74.3	71	60.3	24	86.6
Floyd	160	205	58	71.7	73	64.4	24	88.3
All-pole	290	311	86	72.3	115	63.0	37	88.1
WDF(1)	40	506	147	70.9	204	59.7	71	86.0
WDF(2)	120	140	43	69.3	58	58.6	21	85.0
2D(1)	340	402	124	69.2	165	59.0	51	87.3
2D(2)	40	501	158	68.5	215	57.1	77	84.6
MDFG1	80	102	36	64.7	40	60.8	12	88.2
MDFG2	80	124	31	75.0	48	61.3	15	87.9
Average Reduction (%)				70.5		60.3		86.5

TABLE II

THE COMPARISON OF PERFORMANCE WITH FOUR METHODS ON VARIOUS BENCHMARKS FOR 8 CORES.

algorithm. Table II shows our experimental results. Column "Bench." stands for the benchmarks we used in the experiments. Column "N." represents the number of nodes of each filter benchmark. We have unfolded 10 times for each benchmark in order to get large enough number of nodes. Entries " M-1" to " M-4" represent the four methods implemented in the experiments. Column "T" represents the execution time obtained from six different methods. Column "% M1" after each method represents the percentage of reduction in execution time, compared to the Method 1 respectively. The last row shows the average reduction.

The results show that our EALPM can significantly improve the performance for multi-core architectures. For example, with 8 cores, EALPM shows an average 86.5% reduction of total execution time, compared with the approach without using parallelism (M-1). Our further experiments show that with more cores available, the reduction ratio for the total execution time has increased.

V. CONCLUSION

In this paper, we studied the loop parallelism and voltage assignment problem to maximize loop parallelism with energy consideration on multidimensional nested loops. We proposed a highly efficient algorithm, EALPM (*Energy-Aware Loop Parallelism Maximization*) for applications with loops. By combining loop transformation and voltage assignment, our algorithm improves both the performance (with parallelism) and energy-saving (with voltage scaling) for multidimensional DSP applications on heterogeneous (also includes homogeneous) multi-core platforms. A wide range of benchmarks have been tested in the experiments. The experimental results showed that our algorithm significantly improves both performance and energy-saving for applications with nested loops.

VI. ACKNOWLEDGEMENT

This work is partially supported by NSFC 61071061 and University of Kentucky start up fund; Research Fund of the State Key Laboratory of Software Development Environment under Grant No. BUAA SKLSDE-2010ZX-13; the NSF Grants CNS-0915762 (CSR); CCF-0845257

(CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), Auburn startup grant and Intel gift (Number 2005-04-070); NSFC 60874050, NCET-10-0692, Zhejiang NSFC R1100234 and Z1090423, ZPED Z200909334, FRFCU 2009QNA4012, and FAS 20102076002.

References

[1] X. Tang and G. R. Gao, "Automatically partitioning threads for multithreaded architectures," *Journal of Parallel and Distributed Computing*, vol. 58(2), 1999.

[2] D. Liu, Z. Shao, M. Wang, M. Guo, and J. Xue, "Optimal loop parallelization for maximizing iteration-level parallelism," in *ACM CASES*, Oct. 2009.

[3] A. Aiken and A. Nicolau, "Optimal loop parallelization," *ACM SIGPLAN Notices*, vol. 23(7), 1988.

[4] R. Allen and K. Kennedy, *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann, 2001.

[5] M. Kandemir, N. Vijaykrishnan, M. Irwin, and W. Ye, "Influence of compiler optimizations on system power," in *DAC*, 2000.

[6] G. Chen, M. Kandemir, and M. Karakoy, "Compiler support for voltage islands," in *IEEE International SOC Conference*, Austin, TX, Sep. 2006.

[7] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, 1991.

[8] N. Passos and E. H.-M. Sha, "Achieving full parallelism using multi-dimensional retiming," *IEEE Trans. Parallel and Distributed Systems*, vol. 7(11), Nov. 1996.

[9] L.-F. Chao, A. LaPaugh, and E. H.-M. Sha, "Rotation scheduling: A loop pipelining algorithm," *IEEE Trans. on Computer-Aided Design*, vol. 16, Mar. 1997.

[10] M. Fleischmann, "Crusoe power management– reducing the operating power with longrun," in *12th HOT CHIPS Symp.*, Aug. 2000.

[11] L. T. Clark, "Circuit design of $xscale^{TM}$ microprocessors," in *Symp. on VLSI Circuits, Short Course on Physical Design for Low-Power and High-Performance Microprocessor Circuits*, June 2001.

[12] M. Qiu, E. H.-M. Sha, M. Liu, M. Lin, S. Hua, and L. T. Yang, "Energy minimization with loop fusion and multi-functional-unit scheduling for multidimensional DSP," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 68, no. 4, pp. 443–455, Apr. 2008.

[13] M. Qiu, Z. Jia, C. Xue, Z. Shao, and E. H.-M. Sha, "Voltage assignment with guaranteed probability satisfying timing constraint for real-time multiproceesor DSP," *Journal of VLSI Signal Processing Systems (JVLSI)*, vol. 46, no. 1, Jan. 2007.

[14] A. Iyer and D. Marculescu, "Power-performance evaluation of globally asynchronous, locally synchronous processors," in *29th Intl. Symp. on Computer Architecture*, May 2002.

[15] G. Semeraro, D. Albonesi, S. Dropsho, G. Magklis, S. Dwarkadas, and M. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *IEEE MICRO*, Nov. 2002.

[16] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *IEEE HPCA*, Feb. 2002.

[17] M. Qiu, L. T. Yang, Z. Shao, and E. H.-M. Sha, "Dynamic and leakage energy minimization with soft real-time loop scheduling and voltage assignment," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 18, no. 3, pp. 501–504, Mar. 2010.

[18] M. Liu, Q. Zhuge, Z. Shao, and E. H.-M. Sha, "General loop fusion technique for nested loops considering timing and code size," in *CASES*, 2004.

[19] M. Liu, Q. Zhuge, Z. Shao, C. Xue, M. Qiu, and E. H.-M. Sha, "Loop distribution and fusion with timing and code size optimization for embedded dsps," in *L.T. Yang et al. (Eds.): EUC 2005, LNCS 3824*, 2005, pp. 121–130.

[20] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power i/o," *IEEE Trans. on VLSI Syst.*, vol. 3, no. 1, March 1995.

[21] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *DAC*, 2002.

[22] D. Shin, J. Kim, and S. Lee, "Low-energy intra-task voltage scheduling using static timing analysis," in *DAC*, 2001.

[23] H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C.-H. Hsu, and U. Kremer, "Energy-conscious compilation based on voltage scaling," in *LCTES'02*, June 2002.

[24] T. Sakurai and A. R. Newton, "Alpha-power law mosfet model and its application to cmos inverter delay and other formulas," *IEEE J. Solid-State Circuits*, vol. SC-25, no. 2, 1990.

[25] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power cmos digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, April 1992.

[26] D. Lackey, P. Zuchowski, T. Bednar, D. Stout, S. Gould, and J. Cohn, "Managing power and performance for system-on-chip designs using voltage islands," in *IEEE ICCAD*, Nov. 2002.

[27] J. Huz, Y. Shinx, N. Dhanwaday, and R. Marculescuz, "Architecting voltage islands in core-based system-on-a-chip designs," in *ACM ISLPED*, Newport Beach, CA, Aug. 2004.

[28] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *IEEE/ACM international conference on Computer-aided design (ICCAD)*, 2002, pp. 721–725.