Performance and Energy Efficiency Analysis of Data Reuse Transformation Methodology on Multicore Processor

Abdullah Al Hasib[1], Per Gunnar Kjeldsberg[2], and Lasse Natvig[1]

[1] Department of Computer and Information Science
{abdullah.alhasib, lasse.natvig}@idi.ntnu.no
[2] Department of Electronics and Telecommunications
Norwegian University of Science and Technology, Trondheim, NO-7491, Norway

Abstract. Memory latency and energy efficiency are two key constraints to high performance computing systems. Data reuse transformations aim at reducing memory latency by exploiting temporal locality in data accesses. Simultaneously, modern multicore processors provide the opportunity of improving performance with reduced energy dissipation through parallelization. In this paper, we investigate to what extent data reuse transformations in combination with a parallel programming model in a multicore processor can meet the challenges of memory latency and energy efficiency constraints. As a test case, a "full-search motion estimation" kernel is run on the Intel® Core™ i7-2600 processor. Energy Delay Product (EDP) is used as a metric to compare energy efficiencies. Achieved results show that performance and energy efficiency can be improved by a factor of more than 6 and 15, respectively, by exploiting a data reuse transformation methodology and parallel programming model in a multicore system.

Keywords: Performance, energy efficiency, data reuse transformation methodology, parallel programming.

1 Introduction

The rapid growth of microprocessor performance for the last two decades has provided us the opportunity to solve increasingly advanced problems that require very large scale computations. However, memory latency has not been improved at a comparable rate and has become a major limiting factor for system performance. System performance is further impeded by battery capacity for handheld devices and by heat dissipation constraints for high performance processor designs [1]. Therefore, improvement of energy efficiency and memory latency has now become a major concern in contemporary computer architectures.

For data-dominated applications such as multimedia algorithms, Data Transfer and Storage Exploration (DTSE) offers a complete methodology for obtaining and evaluating a set of data reuse transformations in terms of memory energy [2].

The fundamental idea behind data reuse transformations is to move the data accesses from background memories to smaller and less energy intensive foreground memory blocks in a systematic way. This approach eventually results in significant energy savings.

In this paper, we present a technique that combines a data reuse transformation methodology with a parallel programming model to improve energy efficiency. We evaluate the performance and energy efficiency of our combined technique on a quad-core processor. We have used a "full-search motion estimation" algorithm as our test application.

This paper is organized as follows: Section 2 describes related work. Section 3 presents our methodology to improve energy efficiency and 4 illustrates our methodology using the *motion estimation* algorithm. Section 5 presents and discusses our results. Finally, we conclude the paper in Section 6.

2 Related Work

Research on data reuse transformation methodologies for multimedia applications has been actively performed for the last few decades and has led to numerous approaches to improve memory latency. Wuytack *et al.* [2] present a formalized methodology for data reuse exploration to reduce memory energy consumption by exploiting temporal locality of memory accesses using an optimized custom memory hierarchy. It is taken further and oriented towards predefined memory organizations in [3]. In [4, 5], the authors evaluated the effect of data reuse decisions on power, performance and area in embedded processing systems. The effect of data reuse transformations on a general purpose processor has been explored in [6]. In [7], the authors presented the effect of data reuse transformations on multimedia applications on application specific processors. The research described so far emphasizes on single-core systems and relies on simulation based modeling when energy efficiency is estimated.

In [8], Kalva *et al.* presented the effect of parallel programming on multimedia applications but it lacks energy efficiency analysis. In [9], Chen *et al.* presented different optimization opportunities of the Fast Fourier Transform (FFT) to achieve a high performance implementation on IBM Cyclops-64 chip architecture. In [10], Zhang *et al.* presented an inter-core data reuse technique to exploit all the available cores to boost overall application performance. These earlier studies on parallel architectures emphasize performance rather than energy efficiency. In contrast, we have performed energy efficiency analysis on a state-of-the-art multicore processor with four cores and have used Model Specific Registers of the processor to accurately measure the consumed energy.

The previous work closest to ours is done by Marchal *et al.* [11]. It presents an approach for integrated task-scheduling and data-assignment for reducing SDRAM costs for multi-threaded applications. It does not couple the data reuse analysis with a parallel programming model the way we do here, however.

3 Energy Efficient Methodology for Multicore Processor

In this paper, we present an approach that combines the concepts from a *data reuse transformation* methodology with a *parallel programming* model to get better performance and energy efficiency.

Data Reuse Transformation: The fundamental concept of a data reuse transformation is to optimize an application and/or introduce a custom memory hierarchy to exploit the temporal locality of data accesses [2]. The memory hierarchy consists of layers of gradually smaller memories. The application code is optimized so that data that is accessed multiple times, i.e., has a high reuse factor, is copied from larger to smaller memories closer to the data path. Unless the memory hierarchy is fixed, the size and interconnect of each layer can also be optimized. For data-intensive applications, this approach gives significant energy savings since smaller memories consume less energy per access.

Parallel Programming: Multicore processors can achieve higher performance with lower energy consumption compared to a uniprocessor system. It is, however, a challenging job to develop efficient parallel applications that exploit the advantages of hardware parallelism. Different parallel programming models have been developed that can speed up applications when multiple threads or multiple processes are used [12]. At this level, parallel programs can be written using multi threaded programming and using explicit threading supported by the operating system or using programming frameworks such as OpenMP [13].

In our approach, initially we have applied different possible data reuse transformations described in [2] to optimize energy efficiency for a given algorithm. The first step identifies data sets that are reused multiple times within a short period of time, i.e., *copy candidates*. For each of the identified data sets, a copy to a smaller memory can be introduced so that data is accessed using less energy. Based on a cost trade off with extra copying of data and chip area overhead, a hierarchical memory organization is generated and an optimized set of *copy candidates* are utilized. After data reuse optimization, we develop a parallel algorithm based on the optimized solution.

4 Demonstrator Application: Motion Estimation Kernel

We have used a "full-search motion estimation" algorithm to evaluate the performance and energy efficiency of our combined approach.

4.1 Sequential Unoptimized Motion Estimation Algorithm

Motion Estimation (ME) is a core part of different video compression algorithms. Block-based ME algorithms involve finding the candidate block within a specified search area in a *reference frame* that is most similar to the current block in the *current frame*. A "full-search motion estimation" algorithm performs an exhaustive search over the entire search region to find the optimal solution.

Algorithm 1 Sequential Unoptimized Motion Estimation Algorithm [14]

```
1: for g=0; g<H/n; g++ do
2:     for h=0; h<W/n; h++ do
3:         Δopt[g][h] = +∞
4:         for i=-m; i<m; i++ do
5:             for j=-m; j<m; j++ do
6:                 Δ = 0
7:                 for k=0; k<n; k++ do
8:                     for l=0; l<n; l++ do
9:                         Δ+= abs(Cur[g×n+k][h×n+l] - Ref[g×n+i+k][h×n+j+l])
10:                    end for
11:                end for
12:                Δopt[g][h] = min(Δ, Δopt[g][h])
13:            end for
14:        end for
15:    end for
16: end for
```

This process is computationally intensive and costs about 80% of the encoding time [8]. Therefore, we have chosen it as a test application in our experiment.

Full-search motion estimation is illustrated in Algorithm 1. The implementation of the ME algorithm consists of a number of nested loops. The basic operation at the innermost loop consists of an accumulation of pixel differences, while the basic operation two levels higher in the loop hierarchy consists of the selection of the new minimum. This algorithm is a sequential implementation without exploiting any data reuse transformation technique and referred as *sequential unoptimized* solution in this paper. For our experiment, we have used parameters of the QCIF format *(W=176, H=144, m=n=8)* [14].

4.2 ME Optimization Using Data Reuse Transformations

We have followed a systematic approach presented in [2] to transform the Basic ME Algorithm into an optimized solution that maps selected copies of data on a memory hierarchy to exploit temporal locality. Fig. 1 presents different possible transformations for the ME algorithm.

Each branch in the *copy candidate* tree corresponds to a potential memory hierarchy for different data-reuse transformations. Dashed lines in the figure indicate levels of the hierarchy. Each rectangle in the hierarchy corresponds to a *copy candidate*, i.e., a block of data that can benefit from being accessed multiple times from the given hierarchy level. Each *copy candidate* is annotated with its size. The highlighted path in the figure indicates a *3 layer memory* hierarchy for data reuse transformations on the *reference frame*. The hierarchy is comprised of a $H \times W$ block for the full frame, a $(2m+n-1) \times (2m+n-1)$ block and a $(2m+n-1) \times n$ block for smaller *copy candidates*. In addition, a *2 layer memory* hierarchy for the *current frame* with a $H \times W$ frame memory and a $n \times n$ *copy candidate* is also introduced.

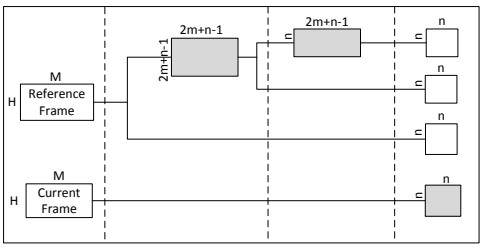


Fig. 1: Copy candidate tree for data reuse decision for Motion Estimation Algorithm. The process of constructing such copy candidate tree is explained in[2]

To evaluate the performance and energy efficiency of the different data reuse transformations presented in Fig. 1, the basic ME algorithm has been modified into different versions to exploit different possible transformations. Achieved performance and energy efficiency of all transformed algorithms are then measured and compared. The transformation that provides the best energy efficiency is converted to a parallel program. Algorithm 2 depicts an example of the transformed ME algorithm with two layers. The transformed algorithm introduces a smaller memory block *(Buffer)* to which the *copy candidate* is copied.

4.3 Parallel Optimized Motion Estimation Algorithm

The Motion Estimation algorithm also exhibits important properties of data parallelism. In QCIF format, a video frame is comprised of a fixed number of macro blocks (8×8 non-overlapping blocks). Prediction for a given block is determined by finding a block in a given search range of the *reference frame* that is closest to the current block. For each macro block (MB), this estimation can be done in parallel. Algorithm 2 represents our parallel ME algorithm. We have made our *2 layer* ME algorithm parallel by adding the *#pragma omp parallel for* directive of the OpenMP programming model [13] at the outermost *for* loop. This directive will instruct the compiler to distribute the work done in the for-loop immediately following the directive among all processors (cores) of the system. Variables *Ref, Cur* and Δ_{opt} are shared among the threads whereas *(h, Buffer)* are private to each thread. Note that threads should be properly synchronized while computing the minimum Δ_{opt}. Therefore a *#pragma omp critical* directive is used to ensure that Δ_{opt} is accessed by a single thread at a time. We have also set the *GOMP_CPU_AFFINITY* environment variable to bind each thread, i.e., each instance of the *for-loop*, to a specific core.

4.4 System Architecture and Energy Measurement

System Architecture: In our experiment, we have used the Intel® Core™ i7-2600 processor which consists of four physical cores. It supports Hyper-Threading allowing it to simultaneously process up to 8 threads, i.e., 2 threads per core.

Algorithm 2 Parallel Optimized Motion Estimation Algorithm

```
1: #pragma omp parallel for shared(Ref, Cur, Δopt) private(h, Buffer)
2: for g=0; g<H/n; g++ do
3:     for h=0; h<W/n; h++ do
4:         for k=0; k<2m+n-1; k++ do
5:             for l=0; l<2m+n-1; l++ do
6:                 Buffer[k][l]=Ref[g×n-m+k][h×n-m+l]
7:             end for
8:         end for
9:         Δopt[g][h] = +∞
10:        for i=0; i<2m-1; i++ do
11:            for j=0; j<2m-1; j++ do
12:                Δ = 0
13:                for k=0; k<n; k++ do
14:                    for l=0; l<n; l++ do
15:                        Δ += abs(Cur[g×n+k][h×n+l]- Buffer[i+k][j+l])
16:                    end for
17:                end for
18:                #pragma omp critical
19:                Δopt[g][h] = min(Δ, Δopt[g][h])
20:            end for
21:        end for
22:    end for
23: end for
```

The memory hierarchy consists of a 32 KB Level-1 cache, a 256 KB Level-2 cache and an 8192 KB Level-3 cache. Level-1 and Level-2 caches are private to each core while the Level-3 cache is shared among the cores. Note that this is a memory hierarchy with a fixed number of levels and sizes, typical for a standard processor. This is different from the assumption in [2], where an application specific memory hierarchy is assumed. The base clock speed of the processor is 3.4 GHz, but it can go as high as 3.8 GHz when Turbo Boost is enabled [15].

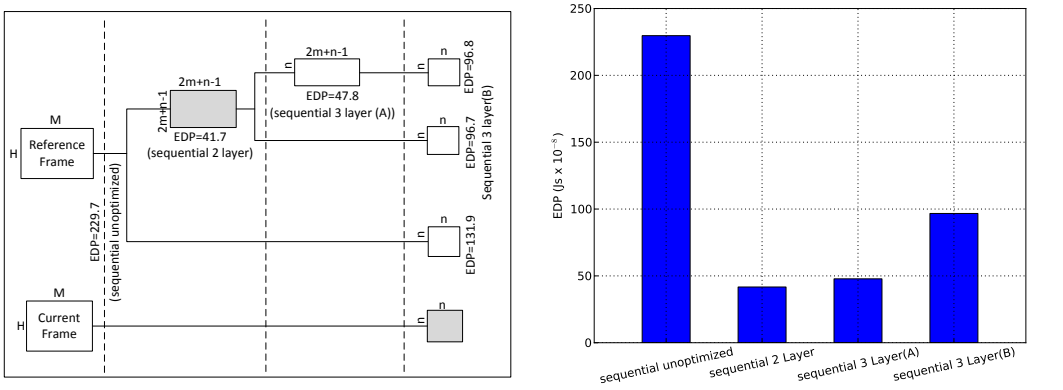
Energy Measurement Policy: We read the non-architectural Model Specific Registers of the processor to estimate on-chip energy consumption [15]. The *MSR_PP0_ENERGY_STATUS* register gives us aggregate energy consumed by the cores as well as caches. We read this register at a fixed core frequency (3.4 GHz) and process the raw data to compute energy efficiency.

Energy Efficiency Metric: We report energy efficiency in terms of the Energy-Delay-Product (EDP) metric [16]. Measured units for Energy and Delay are *Joule(J)* and *second(s)* respectively. Therefore, the unit of the EDP metric is *Js*. Generally, the lower the EDP, the better the energy efficiency is.

OS and Compiler Parameters: We execute our experiment on OpenSuse 11.4 (x86 64) running Linux kernel 2.6.37.6. The parallel application is compiled using the *gcc* compiler with *-fopenmp* flag and optimized with *-O3* flag.

5 Results and Discussion

Energy Efficiency Evaluation of Sequential ME Algorithm: Different data reuse transformations of the sequential ME algorithm and their corresponding energy efficiencies are presented in Fig. 2.



(a) EDP (Js$\times 10^{-8}$) of different transformations (b) EDP is minimum at 2-layer transformation

Fig. 2: Data reuse transformations and their energy efficiencies measured in EDP

Fig. 2a shows that energy efficiency is improved significantly due to data reuse transformation techniques despite of the fact that such transformations introduce both area and computational overheads. For instance, the *sequential 2 layer* transformation introduces a *(2m+n-1)×(2m+n-1)* block buffer for the *reference frame* and a *(n×n)* block buffer for the *current frame* and these additional buffers cost 2372 Bytes of area overhead. The computational and energy overhead to copy the *copy candidates* into the buffer are 0.31 microsecond and 6.67 millijoule, respectively. Therefore, in terms of EDP, the overhead is approximately 0.207×10^{-8} Js for each *new frame*. Despite these overheads, we have observed that achieved EDP for the complete handling of one *new frame* is 229.7×10^{-8} Js for the *sequential unoptimized* ME Algorithm whereas the EDP of the *sequential 2 layer* transformation is 41.7×10^{-8} Js. This improvement is attributed to the use of smaller buffers since a block of *(2m+n-1)×(2m+n-1)* integer-numbers corresponds to 2116 (23×23×4) bytes which is less than the Level-1 cache size in our system. As a result, the buffer can be brought into the Level-1 cache during the computation which significantly reduces the cost of expensive memory accesses and improves performance as well as energy efficiency.

An important observation from Fig. 2b is that the efficiency is at a peak with a *2 layer memory* hierarchy and it degrades with the introduction of any additional layers of smaller memory blocks. Two factors that contribute to this

result are: (i) Additional memory layers also introduce additional area and computational overheads (ii) Smaller data blocks are copied into relatively larger cache-blocks due to the fixed-sized caches, which ultimately negate the advantage of using additional memory layers.

Energy Efficiency Evaluation of Parallel ME Algorithm: To maximize the energy efficiency, we have converted the optimized ME algorithm that uses a *2 layer memory hierarchy* into a parallel one by using the OpenMP programming model and executed it on our system with a varying number of threads. Fig. 3 presents the obtained result.

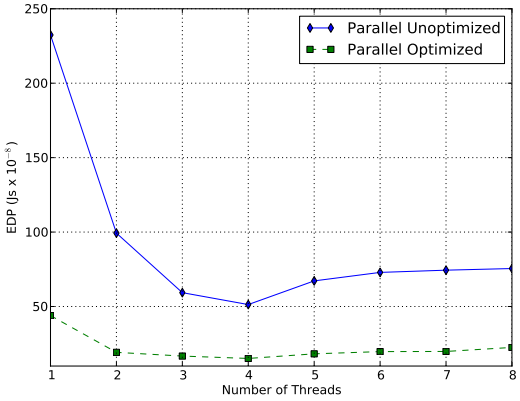


Fig. 3: Improved energy efficiency using optimized parallel ME algorithm

Fig. 3 implies that parallel programming improves energy efficiency of both optimized (that exploits data reuse transformation methodology) and unoptimized solutions (not using data reuse transformations). We can see that EDP values drop rapidly with increasing number of threads and reach their minimum when 4 threads are used. Since the Intel® Core™ i7-2600 processor consists of 4 physical cores which are shared among the threads in Hyper-Threading mode, cache pollution causes the *parallel unoptimized* solution to increase the EDP values with the increasing number of threads. In contrast to the unoptimized solution, the optimized solution exhibits better cache behavior due to the use of smaller memory blocks. Hence, EDP remains almost constant during the Hyper-Threading mode.

Table 1 presents a summary of our results which reveal that data reuse transformations significantly improve energy efficiency and that the *parallel optimized* solution is the most energy efficient transformation for ME algorithm. Normalized EDP values (with respect to *optimized parallel solution*) in the Table indicate that, *sequential optimized* and *sequential unoptimized* solutions are energy

in-efficient by a factor of 2.7 and 15.1, respectively. The execution time for performing ME on one complete *new frame* is improved with a factor of 6.5 going from *sequential unoptimized* to *parallel optimized*.

Table 1: Results of different data reuse transformations

Version	**Execution Time** Second$\times 10^{-6}$	**Energy** Joule$\times 10^{-3}$	**Energy Efficiency (EDP)** Js$\times 10^{-8}$	**Normalized EDP**
Sequential Unoptimized	10.9	210.7	229.7	15.1
Sequential 2 Layer	4.3	97.0	41.7	2.7
Sequential 3 Layer	4.6	104.1	47.8	3.1
Parallel Unoptimized	3.2	161.4	51.6	3.4
Parallel Optimized	1.7	89.7	15.2	1.0

In contrast to our results, in which we have obtained the best energy efficiency by using a *2 layer memory* hierarchy, Wuytack *et al.* in [2] have shown that a *3 layer memory* hierarchy is the most energy efficient scheme for the ME algorithm. However, our experiments differ from their as follows: First, we have experimented on a processor with a memory hierarchy of fixed sized cache-blocks. The *copy candidates* are hence mapped to a portion of these fixed-size system caches, and consequently our measurements consider the energy consumed by both used and idle cache lines. Wuytack *et al.* did their experiment in a simulation environment that created a hierarchy of memory blocks that perfectly fit the data blocks. Therefore, extra energy consumption due to unused cache area is avoided. To avoid extra energy consumption in our experiment, we would need to have an execution platform using a concept like *drowsy cache* [17] that powers down unused parts of the cache. This would give more comparable results between the two methods. It is not available in the Core$^{\text{TM}}$ i7 processor, however. Second, we have measured energy efficiency of the complete program rather than a part of the program that deals with data transfer. Third, we have measured on-chip memory and core energy consumption rather than considering only memory energy consumption. Fourth, their simulation environment assumes that data can be directly copied from a low-level hierarchy to a high-level hierarchy bypassing any intermediate layer. This is not possible in our system.

6 Conclusion

In this paper, we have investigated performance and energy efficiency effects of applying data-reuse transformations on a multicore processor running a motion estimation algorithm. We have shown that for a sequential *Motion Estimation* kernel, energy efficiency can be improved up to 5.5 times by using appropriate data-reuse transformation techniques, which can be further extended to 15.1 times by incorporating the OpenMP parallel programing model. We have also shown that Hyper-Threading degrades both performance and energy efficiency of the unoptimized solution. This gives clear indications that a data reuse transformation methodology in combination with a parallel programming model can

significantly save energy as well as improve performance of this type of applications running on multicore processors.

References

1. Albers, S.: Energy-Efficient Algorithms. Communications of the ACM **53**(5) (May 2011) 86–96
2. Wuytack, S., Diguet, J.P., Catthoor, F., et al.: Formalized Methodology for Data Reuse Exploration for Low-Power Hierarchical Memory Mappings. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **6**(4) (Dec. 1998) 529–537
3. Catthoor, F., Danckaert, K., Kulkarni, K., et al.: Data Access and Storage Management for Embedded Programmable Processors. Kluwer Academic Publishers, Dordrecht, The Netherlands (2002)
4. Catthoor, F., Wuytack, S., de Greef, G., et al.: Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design. Kluwer Academic Publishers, Norwell, MA, USA (1998)
5. Zervas, N.D., Masselos, K., Goutis, C.E.: Data-Reuse Exploration for Low-Power Realization of Multimedia Applications on Embedded Cores. In: Proc. 9th International Workshop on Power and Timing Modeling, Optimization and Simulation. PATMOS'99 (1999) 71–80
6. Chatzigeorgiou, A., Chatzigeorgiou, E., Kougia, S., et al.: Evaluating the Effect of Data-Reuse Transformations on Processor Power Consumption (2001)
7. Vassiliadis, N., Chormoviti, A., Kavvadias, N., et al.: The Effect of Data-Reuse Transformations on Multimedia Applications for Application Specific Processors. In: Proc. Intelligent Data Acquisition and Advanced Computing Systems Technology and Applications. IDAACS'05 (Sep. 2005) 179–182
8. Kalva, H., Colic, A., Garcia, A., et al.: Parallel Programming for Multimedia Applications. Multimedia Tools and Applications **51**(2) (Jan. 2011) 801–818
9. Chen, L., Hu, Z., Lin, J., et al.: Optimizing the Fast Fourier Transform on a Multicore Architectures. In: Proc. Parallel and Distributed Processing Symposium. IPDPS'07 (Mar. 2007) 1–8
10. Zhang, Y., Kandemir, M., Yemliha, T.: Studying Inter-core Data Reuse in Multicores. In: Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems. SIGMETRICS '11 (2011) 25–36
11. Marchal, P., Catthoor, F., Bruni, D., et al.: Integrated Task Scheduling and Data Assignment for SDRAMs in Dynamic Applications. IEEE Design & Test of Computers **21**(5) (Sep. 2004) 378–387
12. Podobas, A., Brorsson, M., Faxén, K.F.: A Comparison of some recent Task-based Parallel Programming Models. In: Proc. 3rd Workshop on Programmability Issues for Multi-Core Computers, Pisa, Italy (Jan. 2010)
13. OpenMP Architecture Review Board: OpenMP Application Program Interface. http://www.openmp.org/mp-documents/OpenMP3.1.pdf (Jul. 2011)
14. Komarek, T., Pirsch, P.: Array Architectures for Block Matching Algorithms. IEEE Transactions on Circuits and Systems **36**(10) (Oct. 1989) 1301–1308
15. Intel: Intel 64 and IA-32 Architectures Software Developer's Manual. (2011)
16. Rivoire, S., Shah, M.A., Ranganathan, P., et al.: Models and Metrics to Enable Energy-Efficiency Optimizations. Computer **40**(12) (Dec. 2007) 39–48
17. Flautner, K., Kim, N.S., Martin, S., et al.: Drowsy Caches: Simple Techniques for Reducing Leakage Power. In: Proc. 29th Annual International Symposium on Computer Architecture. ISCA '02, Washington, DC, USA (2002) 148–157