Energy Efficiency of Multithreaded WZ Factorization with the use of OpenMP and OpenACC on CPU and GPU

Beata Bylina[0000−0002−1327−9747] and Jarosław Bylina[0000−0002−0319−2525]

Institute of Computer Science
Marie Curie-Skłodowska University
Lublin, Poland
{beata.bylina,jaroslaw.bylina}@umcs.pl

Abstract. Energii efficiency research aims to optimize the use of computing resources by minimizing energy consumption and increasing computational efficiency. This article explores the effect of the directive-based parallel programming model on energy efficiency for the multithreaded WZ factorization on multi-core central processing units (CPUs) and multi-core Graphics Processing Units (GPUs). Implementations of the multithreaded WZ factorization (both the basic and its variant optimized by strip-mining vectorization) are based on OpenMP and OpenACC. Strip-mining gave clear enhancement in comparison to the basic version. The energy efficiency is much better on GPU than on CPU; and on CPU, the use of OpenMP is more energy-efficient; however, for GPU, OpenACC gives better results.

Keywords: WZ factorization, OpenACC, OpenMP, energy efficiency

1 Introduction

Research focused on enhancing the energy efficiency of algorithms through various programming techniques plays a crucial role in the broader context of sustainable development. In this paper, we will examine energy efficiency using only the ratio of the total number of floating-point operations to the total energy consumption (Flop/J) metric, as presented in paper [11].

The development of multithreaded parallel applications on multi-core architectures can be based on various parallel programming frameworks, including OpenMP (Open Multi-Processing) [9], OpenCL (Open Computing Language) [5], and OpenACC (Open ACCelerators) [8]. Choosing the one that is appropriate for the target context is not easy. OpenMP and OpenACC are two of the most widely used directive-based parallel programming models for parallelization.

The WZ matrix factorization was introduced in 1979 by D.J. Evans and M. Hatzopoulos [2]. It is also known as Quadrant Interlocking Factorization (QIF). The aim of their work was to design a factorization for greater parallelization. The WZ factorization is investigated by variuos researchers, as [1,12].

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

The WZ factorization was chosen because it contains a lot of arithmetic operations that could be performed in parallel and distributed among many cores on both the CPU and GPU. factorization were selected for further research described in this article, namely, the basic one and its variant optimized by strip-mining vectorization, which is the best in terms of performance and energy consumption. In this paper, the focus is on describing the experience of moving ideas and concepts from the OpenMP programming model on multi-core architecture CPU to OpenACC both on multi-core architecture CPU and GPU. In particular, energy efficiency is analyzed and compared for two directive-based programming models on CPU, namely OpenMP and OpenACC, on two platforms (GPU and CPU). Experimental evaluations prove the expected improvement in energy efficiency.

The rest of the paper is organized as follows. In Section 2, contemporary research on parallel programming and its impact on energy consumption was presented. Section 3 details multithreaded implementations of the WZ factorization algorithm, utilizing OpenMP and OpenACC. In Section 4, experimental evaluations of energy efficiency on multi-core CPUs and many-core GPUs are presented. Lastly, Section 5 concludes the paper and outlines future research directions.

2 Related Work

In high-performance computing, energy efficiency via directive-based parallel programming is crucial. Articles [7,10] explore OpenMP's impact on energy efficiency, optimizing execution time and energy consumption. [10] delves into OpenMP's runtime power level adjustments, while [7] studies performance and energy usage in linear algebra kernels. Other works [3,6] compare parallel programming models on multi-core CPUs and GPUs, including OpenMP and OpenACC. The paper [3] assesses CUDA, OpenMP, and OpenACC on Nvidia Tesla V100 GPU for matrix multiplication, highlighting data size's impact on performance. None of them compare OpenMP with OpenACC in dense linear algebra algorithms regarding energy-performance trade-offs.

3 Multithreaded implementations of the WZ factorization

The WZ factorization [12] consists in transforming a nonsingular square matrix $\mathbf{A}$ into a product of two matrices, namely $\mathbf{W}$ and $\mathbf{Z}$. The form of the matrix $\mathbf{W}$ is a butterfly and the for of the matrix $\mathbf{Z}$ — of an hourglass (details can be seen in [1], for example).

The easiest way was to transform OpenMP to OpenACC is to replace `#pragma omp parallel for` enforcing the parallelism of the loops. In OpenACC, parallel directives `#pragma acc parallel loop gang` will be used by the compiler for different architectures (both CPU and GPU). The `loop` directive uses the `gang` clause to instruct the compiler about the level of parallelism. There is no direct equivalent of `#pragma omp simd` in OpenACC. However, a

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

certain substitute of that pragma is `#pragma acc loop vector`. Additionally, we put `independent` which is to notify the compiler that the iterations of the loop are data independent. In the GPU version only, we need to provide the compiler with additional information on how to manage the transfer between the device (GPU) and the host (CPU) and therefore we added `#pragma acc data copy(a) copy(w)`. Lastly, in the OpenMP version on GPU we have to put `#pragma omp target device(0)` — to enforce computations offloading to GPU.

Figure 1 shows the pseudocode (only the inner loop fragment) of the basic version of WZ factorization with the use of OpenMP (blue pragmas) and OpenACC (green pragmas). Purple pragmas are added to make the OpenMP code run on a GPU device.

Therefore, the strip-mining technique is manually applied to the algorithm as shown in Figure 2. A loop in the process of strip-mining is divided into two loops, where the inner one has `BLOCK_SIZE` iterations and the outer one has `n/BLOCK_SIZE` iterations (`n` is the number of iterations in the original loop). The strip-mining alone can have some positive impact on the performance (by easing the automatic vectorization process).

In such a process we improve the temporal and spatial locality of the data. By dividing the data into pieces of `BLOCK_SIZE`, we cause them to fit in cache memory and stay there as long as needed to conduct current computations. This minimizes the frequency of cache memory swaps. The theoretical value of

```
#pragma omp target device(0)
#pragma acc data copy(a[0:n*n]) copy(w[0:n*n])
for(k = 0; k < n/2-1; k++) {
  p = n-k-1; akk = a[k][k]; akp = a[k][p];
  apk = a[p][k]; app = a[p][p]; detinv = 1 / (apk*akp - akk*app);
#pragma omp parallel for
#pragma acc parallel
  {
#pragma acc loop gang
    for(i = k+1; i < p; i++) {
      w[i][k] = (apk*a[i][p] - app*a[i][k]) * detinv;
      w[i][p] = (akp*a[i][k] - akk*a[i][p]) * detinv;
// INNER LOOP
#pragma omp simd
#pragma acc loop vector independent
      for(j = k+1; j < p; j++)
        a[i][j] = a[i][j] - w[i][k]*a[k][j] - w[i][p]*a[p][j]; } } }
```

Fig. 1. The version of the basic algorithm with the use of OpenMP and OpenACC — pseudocode.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

BLOCK_SIZE can be estimated from: $(\texttt{BLOCK\_SIZE}^2 + 4 \cdot \texttt{BLOCK\_SIZE}) \cdot s \leq M$ where M is the size (in bytes) of the cache memory considered and s is the size (also in bytes) of one floating-point number. Thus we can see that it should be satisfied (for the L2 cache size $M = 1024$ kB and the size of the double precision $s = 8$ B) if $\texttt{BLOCK\_SIZE} \leq 256$ (we use only divisors of the size of the matrix).

4 Numerical experiments

We tested two types of the WZ factorization algorithm: `basic`, which is the algorithm presented in Figure 1, and `sm-b`, which refers to the strip-mining algorithms depicted in Figure 2.

All the implementations were tested on randomly generated dense matrices which had the WZ factorization with no pivoting needed. The sizes of the matrices were: 8192, 16384, 32768 (as in [7]). The experiment was conducted on a CPU Intel(R) Xeon(R) Gold 5218R with codename Cascade Lake-SP, featuring 40 cores (20 per socket), 80 threads (2 per core), and a SIMD register size of 512 bits. During the tests, Intel ICC (version 2021.5.0) was used, with the following compiler options: `-O3 -qopenmp -xHOST -ipo -no-prec-div -fp-model fast=2 -qopt-zmm-usage=high`. The `-xHOST` option generates optimized code based on the Intel(R) Xeon(R) Gold processor capabilities, but does not utilize 512-bit ZMM registers. To utilize AVX2 instructions on ZMM registers, the `-qopt-zmm-usage=high` option is added. To analyze the impact of all versions of the algorithm on energy consumption on CPU, we used measurements from the RAPL (Running Average Power Limit) interface [4].

The experiment was also conducted on a Tesla V100S-PCIE-32GB GPU. It features a 1370 MHz core clock, 32 GB memory, and 5120 CUDA cores. The compiler used during testing was `nvc`. GPU power sensors were monitored using the NVIDIA System Management Interface (`nvidia-smi`), based on the NVIDIA Management Library (NVML).

```
// INNER LOOP
      start = RDTTNM(k+1, BLOCK_SIZE);
      for(jj = start; jj < p; jj += BLOCK_SIZE) {
        __assume(jj % BLOCK_SIZE == 0);
#pragma omp simd
#pragma acc loop vector independent
        for(j = jj; j < jj+BLOCK_SIZE; ++j)
          a[i][j] = a[i][j] - w[i][k]*a[k][j]- w[i][p]*a[p][j]; } } }
```

Fig. 2. Strip-mining in the basic algorithm with the use of OpenMP and OpenACC — pseudocode (the inner loop only).

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

In Figure 3, we have graphs showing the energy efficiency of the `sm-b` version for block size $b = 64, 128, 256$ on both CPU and GPU, using both OpenMP and OpenACC (NB: different scales were used for GPU and CPU).

Analyzing these data, we can observe that there is no single block size that consistently yields optimal results across all dataset sizes. The experimental results indicate that the energy efficiency of both OpenMP and OpenACC may deteriorate with larger input data sizes. Thus, comprehensive studies are necessary for all dataset sizes.

Contrary to common belief, the best runtime performance does not always correspond to the best energy consumption. Tables 1, 2, and 3 present the time, performance, energy consumption, and energy efficiency of the algorithm versions that performed best during the experiments across all dataset sizes. Analyzing data from these tables, we observe that the GPU version using OpenACC (values in bold) outperforms other versions of the algorithm. Conversely, the worst-performing version is on CPU with OpenACC. Additionally, applying strip-mining to the algorithm results in improvements across time, performance, and energy consumption. For instance, in terms of energy efficiency, the `sm-64` variant on GPU exhibits about a 61% improvement compared to the second-best variant, namely `basic` on GPU for a size of 8192.

Analyzing the collected data, reveals that as matrix size increases, time, performance, energy consumption, and energy efficiency all decline. Similar performance trends are noted in [3].

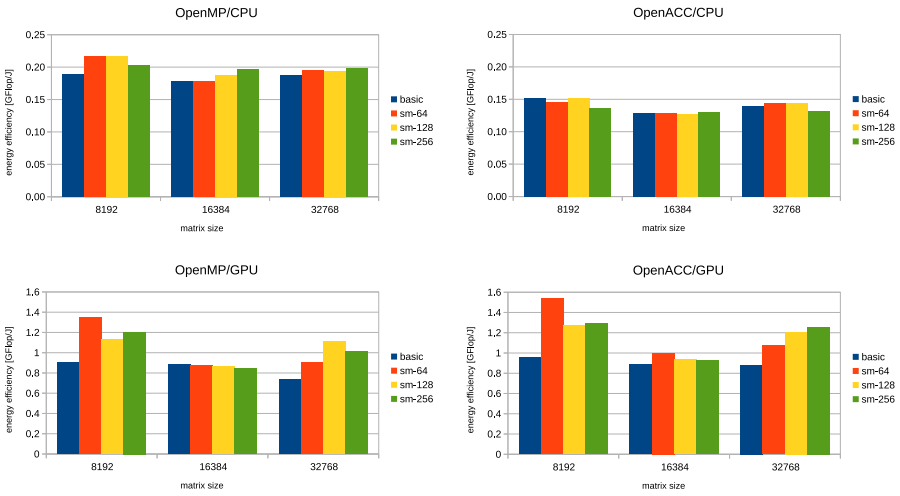

Fig. 3. Energy efficiency for `sm-b` for $b = 64, 128, 256$ on CPU (top) using OpenMP and OpenACC and GPU (bottom) using OpenMP and OpenACC

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

Table 1. Time, Performance, Energy, Energy efficiency for 8192

Versions	Time [s]	Performance [Gflops]	Energy consumption [J]	Energy efficiency [Gflop/J]
`basic` OpenMP CPU	9.64	38.01	1932.87	0.18
`basic` OpenACC CPU	9.99	36.68	2414.23	0.15
`basic` OpenMP GPU	2.58	142,06	405.55	0.90
`basic` OpenACC GPU	2.53	144.86	383.62	0.95
`sm-64` OpenMP CPU	8.67	42.27	1687.53	0.21
`sm-128` OpenACC CPU	10.40	35.22	2419.90	0.15
`sm-64` OpenMP GPU	2.43	150.82	272.77	1.34
`sm-64` OpenACC GPU	2.33	157.29	238.00	**1.53**

Table 2. Time, Performance, Energy, Energy efficiency for 16384

Versions	Time [s]	Performance [Gflops]	Energy consumption [J]	Energy efficiency [Gflop/J]
`basic` OpenMP CPU	75.96	38.59	16512.92	0.17
`basic` OpenACC CPU	92.03	31.86	22845.20	0.12
`basic` OpenMP GPU	19.17	152.95	3312.43	0.88
`basic` OpenACC GPU	17.72	165.46	3306.49	0.88
`sm-256` OpenMP CPU	68.54	42.77	14925.40	0.19
`sm-256` OpenACC CPU	90.13	32.52	22618.68	0.12
`sm-64` OpenMP GPU	13.81	212.31	3352.50	0.87
`sm-64` OpenACC GPU	14.52	201.93	2934.79	**0.99**

Table 3. Time, Performance, Energy, Energy efficiency for 32768

Versions	Time [s]	Performance [Gflops]	Energy consumption [J]	Energy efficiency [Gflop/J]
`basic` OpenMP CPU	582.63	40.26	125098.70	0.19
`basic` OpenACC CPU	681.70	34.41	168578.50	0.14
`basic` OpenMP GPU	138.62	169.21	31869.74	0.74
`basic` OpenACC GPU	140.18	167.33	26657.43	0.88
`sm-256` OpenMP CPU	556.17	42.17	118442.28	0.20
`sm-64` OpenACC CPU	685.11	34.24	162233.58	0.14
`sm-128` OpenMP GPU	140.82	166.57	21172.60	1.12
`sm-256` OpenACC GPU	117.10	200.31	18661.66	**1.26**

4.1 Assembler on CPU

The compiler's task is complex, translating high-level source code into machine code while optimizing processor resources. Assembly code varies across compilers, affecting execution times and energy efficiency. In [6], execution times differ significantly between OpenMP and OpenACC implementations for the same application, likely due to compiler maturity with directives. Our analysis reveals energy consumption and efficiency also depend on compiler maturity. Examining assembler code on CPU with ICC (OpenMP) and nvc (OpenACC), we observe differences in register usage, impacting code vectorization and thus performance and energy efficiency. Longer registers enable better vectorization, but memory bandwidth limitations may mitigate their impact on performance.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

5 Conclusion

This paper develops a multithreaded implementation of the WZ factorization using OpenACC, comparing energy efficiency between OpenMP for multi-core CPUs and OpenACC for both CPUs and GPUs. Strip-mining enhances performance and energy efficiency. Choosing between OpenMP and OpenACC affects time, performance, energy consumption, and efficiency. Porting OpenMP to OpenACC on CPU doesn't improve performance or efficiency, but on GPU, results are better. In general, OpenMP and OpenACC may increase energy efficiency.

The energy efficiency findings in multithreaded WZ factorization can be extended to similar numerical algorithms in computational linear algebra (especially ones, which employ quite regular nested loops), providing valuable insights for developers and researchers.

Future work will focus on developing mathematical models for energy efficiency with OpenMP and OpenACC, and software for optimizing energy efficiency on CPU and GPU automatically.

```
/ICC compiler,  OpenMP, CPU without  -qopt-zmm-usage=high option:
 vmovupd   8(%r15,%rsi,8), %ymm1                          #26.68
 vbroadcastsd (%rdi,%r11,8), %ymm2                        #26.51
 vbroadcastsd (%rdi,%r10,8), %ymm3                        #26.85
 vfnmadd213pd 8(%r12,%rsi,8), %ymm1, %ymm2                #26.68
 vfnmadd132pd 8(%r14,%rsi,8), %ymm2, %ymm3                #26.103
****************************************
/ICC compiler,  OpenMP, CPU with  -qopt-zmm-usage=high option:
 vmovups   8(%r15,%r13,8), %zmm3                          #26.68
 vbroadcastsd (%rdi,%r12,8), %zmm4                        #26.51
 vbroadcastsd (%rdi,%r8,8), %zmm5                         #26.85
 vfnmadd213pd 8(%rsi,%r13,8), %zmm3, %zmm4                #26.68
 vfnmadd132pd 8(%r9,%r13,8), %zmm4, %zmm5                 #26.103
****************************************
/NVC compiler, OpenACC, CPU:
 vmovsd        -8(%r14,%r15,8), %xmm1            # xmm1 = mem[0],zero
 vmovsd        (%r11), %xmm2                     # xmm2 = mem[0],zero
 vfnmadd213sd -8(%r14,%rsi,8), %xmm1, %xmm2
                                        # xmm2 = -(xmm1 * xmm2) + mem
 vmovsd        -8(%r14,%r10,8), %xmm1            # xmm1 = mem[0],zero
 vfnmadd132sd (%r8), %xmm2, %xmm1         # xmm1 = -(xmm1 * mem) + xmm2
```

Fig. 4. The snippet of assembly code for element calculations a[i][j] with the use of OpenMP (ICC compiler — on the top) and OpenACC (NVC — on the bottom) on CPU.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12

References

1. Bylina, B., Bylina, J., Piekarz, M.: Influence of loop transformations on performance and energy consumption of the multithreded WZ factorization. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M., Slezak, D. (eds.) Proceedings of the 17th Conference on Computer Science and Intelligence Systems, FedCSIS 2022, Sofia, Bulgaria, September 4-7, 2022. Annals of Computer Science and Information Systems, vol. 30, pp. 479–488 (2022). https://doi.org/10.15439/2022F251
2. Evans, D.J., Hatzopoulos, M.: A parallel linear system solver. International Journal of Computer Mathematics **7**(3), 227–238 (1979). https://doi.org/10.1080/00207167908803174
3. Khalilov, M., Timofeev, A.: Performance analysis of CUDA, OpenACC and OpenMP programming models on TESLA V100 GPU. Journal of Physics: Conference Series **1740**, 012056 (01 2021). https://doi.org/10.1088/1742-6596/1740/1/012056
4. Khan, K.N., Hirki, M., Niemi, T., Nurminen, J.K., Ou, Z.: Rapl in action: Experiences in using rapl for power measurements. ACM Trans. Model. Perform. Eval. Comput. Syst. **3**(2) (2018). https://doi.org/10.1145/3177754, https://doi.org/10.1145/3177754
5. Khronos Group: OpenCL overview. https://www.khronos.org/opencl/, accessed: April 2024
6. Larrea, V., Budiardja, R., Gayatri, R., Daley, C., Hernandez, O., Joubert, W.: Experiences in porting mini-applications to OpenACC and OpenMP on heterogeneous systems. Concurrency and Computation: Practice and Experience **32** (04 2020). https://doi.org/10.1002/cpe.5780
7. Lima, J.V.F., Raïs, I., Lefèvre, L., Gautier, T.: Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms. The International Journal of High Performance Computing Applications **33**(3), 431–443 (2019). https://doi.org/10.1177/1094342018792079
8. OpenACC Organization: Homepage | OpenACC. https://www.openacc.org/, accessed: April 2024
9. OpenMP ARB: Home — OpenMP. https://www.openmp.org/, accessed: April 2024
10. Shahneous Bari, M.A., Malik, A.M., Qawasmeh, A., Chapman, B.: Performance and energy impact of OpenMP runtime configurations on power constrained systems. Sustainable Computing: Informatics and Systems **23**, 1–12 (2019). https://doi.org/https://doi.org/10.1016/j.suscom.2019.04.002
11. Szustak, L., Wyrzykowski, R., Olas, T., Mele, V.: Correlation of performance optimizations and energy consumption for stencil-based application on Intel Xeon scalable processors. IEEE Transactions on Parallel and Distributed Systems **PP**, 1–1 (05 2020). https://doi.org/10.1109/TPDS.2020.2996314
12. Yalamov, P., Evans, D.: The WZ matrix factorisation method. Parallel Computing **21**(7), 1111–1120 (1995). https://doi.org/https://doi.org/10.1016/0167-8191(94)00088-R, https://www.sciencedirect.com/science/article/pii/016781919400088R

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_12