EG UK Computer Graphics & Visual Computing (2023)
D. Hunter and P. Vangorp (Editors)

Strategies for More Energy Efficient Volume Analysis and Direct Volume Rendering Descriptor Computation

Jacob D. Hauenstein[1] and Timothy S. Newman[1]

[1]The University of Alabama in Huntsville, Huntsville, Alabama, USA

Abstract

A study of energy use (in an x86-class environment) for computation of descriptors used in analysis and in one common scientific visualisation strategy, direct volume rendering (DVR), of volumetric data is presented. Focus is on descriptors used by classic ray-casting-based DVR, including gradients and curvature. Modified computational strategies are considered versus standard approaches on x86. The modified strategies explored include two memory-based ones and four computation-based ones. Use of energy-optimal strategies was able to achieve close to 20% energy savings for gradient descriptor determination. Factor-of-two improvement in energy efficiency for curvature descriptor determination was achieved through these strategies.

CCS Concepts

• ***Human-centered computing*** → *Scientific visualization;* • ***General and reference*** → *Performance;* • ***Hardware*** → ***Power estimation and optimization;*** • ***Software and its engineering*** → *Software performance;*

1. Introduction

Scientific data analysis and visualisation innovator and end-user communities have conducted many studies related to speed and decision quality for a wide range of volumetric dataset analysis and rendering modes. Those studies have focused on impacts of varied descriptors, rendering construction schemes, data staging, etc. (e.g., [ACL17, RPSC99, TBF*22, XTC*22]). In the work here, we tack toward a related performance issue, specifically we consider energy consumption of some popular descriptors used in scientific volume data analysis and rendering, especially for the commonly-used direct volume rendering (DVR) mode of scientific visualisation. Additionally, we consider strategies for reduction of energy consumption involved in computation of these descriptors. Our work here represents an early step toward determining the environmental impact of scientific volume data analysis and visualisation as well as toward finding guidance so that common, core components of such tasks can exhibit a low energy burden, with all the ensuing benefits that accrue from that.

This work here looks at two fundamental components (descriptors) used in volume data analyses and visualisations, namely gradients and curvatures. In DVR, for example, these are used to determine shadings (e.g., [BD21]); end renderings often have been formed using opacity-based composition based on curvature- and/or gradient-based transfer functions that are coupled with the very common illumination schemes from the computer graphics (i.e., Phong or Phong-like illumination schemes). (Many point cloud-based studies have also used curvature-based descriptors, for example in computational fluid simulations [ZSRH19].) A variety of methods for computing gradients and curvature have been proposed in the literature, and some details about such methods are described in Section 2 of this paper. Such descriptors also have volume data analysis applications beyond DVR, such as for registration (e.g., [WRN05]), isosurfacing, etc. – and classic speed and accuracy studies have been done in such domains (with the importance of accuracy receiving increased recent attention, as in [SLYP22]). However, our focus here is on them as components that commonly enable DVR end renderings.

In this paper, Section 2 (§2) provides background on the use of gradients and curvatures in DVR and on the specific gradient estimation and curvature determination methods considered in this work. In it, energy usage considerations, including the process for measuring CPU energy usage (and some details specific to the processor considered in our studies), are also described. Section 3 describes the approaches we explored to reduce energy use for curvature and gradient calculations. Section 4 describes the experimental setup and our experiments to comparatively analyse energy usage. The paper concludes in Section 5.

2. Background

Gradient-based shading continues to be a popular approach in ray-cast direct volume rendering, with some schemes computing gradients during ray marching while others march on pre-computed gradients to reach suitable rendering rates [FZZ*21]. Similarly, curvature-based transfer functions, which require determination of curvature values at each location within the volume, are a popular approach for adding shape-based cues to DVRs [KWTM03].

DOI: 10.2312/cgvc.20231185

Consideration of the energy footprint for direct volume rendering thus requires not just determination of a given renderer's marching or compositing computation's energy usage but also all base descriptor computation's energy usage (even if not all descriptor computation is done at real time during marching).

Focus in this study is on the gradient and curvature descriptors: specifically, energy usage of (1) the commonly used central differencing and associated higher order methods for gradient estimation and (2) two methods for determining curvature values within volume data that have previously been used in conjunction with curvature-based transfer functions for DVR [KWTM03, HN20]. In this section, we first briefly describe some related studies on energy usage before providing background details on the specific gradient estimation and curvature determination methods considered in our studies (reported later).

2.1. CPU energy usage measurement and prior studies thereof

Energy usage of computing has been a focus of a number of prior works, with that focus quite appropriate given a recent report, as relayed by Jay et al. [JOL*23], that approximately 6% of world power use can be ascribed to digital activity – and this percentage is growing. Many such works have introduced schemes to measure and/or reduce CPU energy usage. However, software component energy use is typically unexplored and thus unknown [LV23]. One noted exception is the use of complexity plot visualisations to consider energy cost of matrix multiplication [TWD*13]. We are not aware of any prior works related to energy usage specific to the computation associated with the descriptors used by classic volume dataset visualisation (in particular, gradient estimation and surface curvature determination), though the *accuracy* and/or run *time* of some descriptors (e.g., gradients [WLMN10]) have previously been analysed. However, studies of energy usage have found that energy usage is not always correlated with execution time [TWD*13, HHR*20], and thus those prior gradient and curvature analyses provide limited guidance with respect to the energy usage of such methods.

Many modern processors (including Sandy Bridge and later Intel CPUs) provide a means to internally measure energy usage via the Running Average Power Limit (RAPL) circuitry [WJK*12]. RAPL estimates the CPU's energy usage via a model that considers factors such as hardware counters, leakage, and temperature. In our studies of energy usage (presented later), we have used a RAPL-capable processor in conjunction with the Performance API (PAPI) software [BDG*00] (which supports RAPL measurements).

Even before the inclusion of RAPL, some prior works were able to measure energy usage on processors lacking this feature. For example, Seng et al. [ST03] used 5w resistors inline with the Vcc trace to the processor core to measure code energy use on a Pentium 4 CPU. They found that some C++ code exhibits energy efficiency improvements when compiler-based loop unrolling is used, though they found that some other code does does not. Later work on x86-64 processors found that using very large loop unrolling factors ($>$ 1024) may substantially increase energy consumption [HOK*16].

Energy usage on non-Intel processors has also been studied. For example, Vasilakis [Vas15] have studied instruction level energy usage on ARM Cortex-A7 and Cortex-A15 CPUs. They found that, on such CPUs, instructions use more energy when L1 cache misses occur (compared to when L1 cache hits). They also found that, on such CPUs, floating point instructions typically use more energy (compared to integer instructions), and division instructions have especially high energy usage. They further found that data dependencies between instructions may result in significantly higher energy usage compared to otherwise identical instructions with no data dependencies.

Pereira et al. [PCR*17] have reported that C/C++ code tends to be more energy-efficient than other languages. That finding motivated our use of C/C++ for the work here.

In our own studies of C/C++ code, we have found that code can sometimes increase its energy optimality via directives to inline frequently called functions. Compiler general optimisation level (e.g., *-O1*, *-O2*, or *-O3*) and arithmetic optimisation settings (e.g., gcc's *-ffast-math* option) can also influence some code's energy consumption (e.g., code we examined in some preliminary studies exhibited lowest energy use with the *-O2* or *-O3* optimisation levels combined with *-ffast-math*).

2.2. Methods for gradient estimation

Central differencing (later denoted **central** here) is one gradient estimator explored here. Others include finding differences between adjacent voxels (denoted **inter**); using third order polynomials–with two versions considered here, one centred on either side of the voxel–(denoted **3order_a** and **3order_b**); and using fourth order polynomials (denoted **4order**).

2.3. Methods for surface curvature determination

Two methods for determining surface curvature in volumetric data are explored here. Both methods determine curvature by first estimating the necessary (i.e., first, second, and mixed partial) derivatives and then using these estimated derivatives in conjunction with a standard surface curvature formulation (i.e., the one presented in [KWTM03]).

One method, denoted **OP**, determines surface curvature using derivative estimates obtained via convolutions with kernels sampled from specially constructed orthogonal polynomials [HN20]. **OP** has one parameter: kernel size N. Smaller N's allow for better localisation, while larger kernels are more robust to noise–a prior report [HN20], whose guidance we employ in our studies here, suggested use of $N = 7$ to suitably balance these factors.

The other method, denoted **TE**, determines surface curvature using derivative estimates obtained via convolutions with kernels constructed from the Taylor expansion [KWTM03]. These kernels can be constructed with specific continuity and accuracy properties. In our studies, presented later, we used kernels with C^3 continuity and fourth order accuracy (following [HN20]).

3. Energy optimisation approaches

In this section, we describe the energy optimisation approaches we devised. These approaches take inspiration from observations made

in prior works (i.e., those described in §2.1). Our approaches attempt to reduce energy usage via a variety of strategies, including loop unrolling, mathematics optimisations, general compiler settings, data organisation strategies, etc. First, we describe such strategies for gradient estimation. Then, we describe strategies applied to determining surface curvature in volumetric data.

3.1. Energy saving approaches for gradient estimation

The gradient estimation is a loop-driven process that passes over the volumetric dataset elements. For the higher order polynomial-based gradients, there are somewhat intensive floating-point computations in each step of the process. Thus, one focus was on loop overhead reduction. Reduction of loop overhead could limit (1) exercise of loop prediction logic and (2) certain other loop-carried operations, in turn lowering energy consumption, which may underlie prior findings that loop unrolling sometimes yields energy savings. Another focus was arithmetic computation optimisation. To realise these foci for gradient estimation, we devised three strategies to potentially save energy.

The first strategy, which focuses on loop overhead, is to unroll the loops that perform the passes over the volume for each gradient estimator. It is denoted **Unroll**. Here, such unrolling was performed via gcc/g++'s *-funroll-all-loops* option.

The second strategy explores reducing some floating-point computation overhead. It is denoted **Fast**. It involves avoiding checks for certain floating point computation conditions. It was performed here via the *-ffast-math* compiler option.

The third strategy, which focuses to some extent on both loop and computation overhead, involves using a higher level of compiler optimisation (*-O3*) compared to a baseline implementation (e.g., *-O2*). It is denoted **O3**. This level of optimisation includes two features in particular that may benefit the descriptor computations: it attempts to reuse some computations, especially memory loads and stores between loop iterations, and it attempts to find and eliminate arithmetic subexpressions [Fre23]. (Another motivation: some prior work has found this level of compiler optimisation tends to reduce energy usage.)

3.2. Energy saving approaches for curvature determination

We devised six approaches to potentially save energy when determining surface curvature in volumetric data. They are described here.

The first three are the **Unroll**, **Fast**, and **O3** strategies, all of which were also used as strategies for gradient estimation, as described in the previous section.

Our fourth strategy, denoted **Bespoke**, aims to organise and stage memory accesses in an efficient manner. It was motivated based on prior findings that an increased L1 cache hit rate may result in an energy savings [Vas15]. Both the **OP** and **TE** methods utilise convolution to estimate derivatives. Due to the multidimensionality of volumetric data, convolution, which requires accessing neighbouring data points in every axial direction in order to perform the necessary multiplications / additions, often results in inefficient use of memory as non-contiguous regions of memory are traversed. The **Bespoke** strategy works to increase memory efficiency during convolution by carefully computing and storing intermediate convolution terms across the volume, which allows later avoiding large and/or unpredictable strides that may result in a less efficient use of memory.

Specifically, the **Bespoke** strategy works as follows. First, as a pre-processing step, a list of the unique values in the convolution kernels for the respective curvature method is computed. (E.g., for the **TE** method, a list of all the unique values present in the first and second derivative C^3 continuous, fourth order accurate convolution kernels is manually constructed.) We use k to denote this list of unique values in the kernels (of which there are $\|k\|$ such values, denoted k_1 through k_n). Next, during run time, $\|k\|$ variants of the original input volume (denoted v) are produced, with each of these variants representing the multiplication of each value in v by one of the values in k (requiring $\|k\| \times \|v\|$ extra memory, where $\|v\|$ is the number of values contained in v). (I.e., one variant volume consists of the original volume with each of its entries scaled by k_1; another one consists of entries scaling by k_2; etc.) These multiplications proceed sequentially in memory in order to help ensure cache efficiency. Once the scaled volumes are produced, the convolution is completed via summing relevant entries of these variant volumes.

Our fifth strategy seeks to reduce data dependencies. It is denoted **Pipeline**. It is motivated by prior findings that data dependencies within code often result in increased energy usage [Vas15]. (Moreover, our own studies have found that the increase in energy usage is especially notable when such code contains few additional instructions to be executed between the dependent ones.) To increase the number of instructions available to be executed between dependent instructions, we developed a software pipelined version of the code that computes the final curvature values from the estimated derivatives.

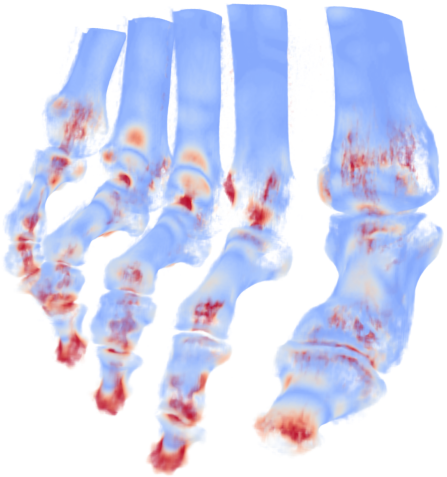
Our sixth strategy seeks to increase data locality by blocking. It is denoted **Blocking**. Increased locality of reference is known to improve cache efficiency, leading to improved computation speed, which motivated us to determine if such increased locality could also reduce energy consumption. It uses a blocked approach to traverse the volume while producing final results. (Our experiments, presented later, tried various block sizes from $8 \times 8 \times 8$ up to $64 \times 64 \times 64$. To save space, **Blocking** in conjunction with a specific blocksize n will be denoted **Blocking**n.)

4. Experimental setup and results

In this section, we report on energy usage of gradient and curvature descriptor computation for a baseline realisation on x86, which is probably the most common CPU environment for most desktop scientific computation. We focus on CPU-based computation here as that is one of the oft-used modes for computing descriptors for DVRs of sensed volume data (such as CT or MR scans of industrial or patient subjects). We also report experiments that evaluate the energy saving strategies that were attempted. Lastly, analyses of accuracy and memory effects are presented.

First, we detail the testing setup. Experiments were performed on

(a) *A slice of the Foot dataset.*



(b) *A DVR of the Foot dataset using curvature-based transfer functions for colour (in a cool-to-warm colourmap, Gaussian curvature).*

Figure 1: *Foot Dataset Renderings*

	central	inter	3order_a	3order_b	4order
Baseline	0.75 J	0.72 J	1.20 J	1.20 J	0.84 J
Fast	2.34%	1.05%	-0.11%	-0.17%	0.57%
Unroll	**-3.81%**	-5.10%	-15.48%	-15.12%	-1.50%
Fast, Unroll	-2.45%	-4.04%	-14.66%	-13.86%	-0.77%
O3	1.13%	-0.35%	0.22%	-0.67%	0.83%
O3, Fast	1.27%	1.82%	0.12%	-0.31%	1.01%
O3, Unroll	-3.77%	**-5.45%**	**-16.15%**	**-15.22%**	**-1.72%**
O3, Unroll, Fast	-3.32%	-3.78%	-14.34%	-14.15%	-0.54%

Table 1: *A selection of **Foot** gradient results.*

	central	inter	3order_a	3order_b	4order
Baseline	0.10 J	0.09 J	0.15 J	0.15 J	0.10 J
Fast	1.05%	-1.16%	-1.53%	-1.43%	0.20%
Unroll	**-5.69%**	-6.21%	**-16.99%**	-17.50%	**-3.85%**
Fast, Unroll	-2.99%	-5.07%	-14.80%	-16.29%	-0.93%
O3	-0.48%	-0.52%	-0.12%	-1.17%	-0.52%
O3, Fast	0.15%	-1.16%	0.74%	-2.72%	0.16%
O3, Unroll	-4.92%	**-8.11%**	-15.92%	**-18.08%**	-2.24%
O3, Unroll, Fast	-4.35%	-4.70%	-15.03%	-16.28%	-2.18%

Table 2: *Selected **Genus3** gradient (energy usage) results.*

a computer equipped with an Intel Core i5-8279U processor and 16GB of RAM (as that matched one used in some prior energy usage reports). The operating system used was a minimal install of Ubuntu Server 22.04.1 AMD64. The CPU governor mode was set to "performance" (using the *cpufreq-set* utility). All experimental code was written in C/C++. Double precision floats were used in all curvature and gradient computations. The code was compiled with *gcc*/*g++* 11.3.0. PAPI was used to measure energy usage (via RAPL). All experiments were run as the root user (to allow access to the energy measurement hardware) on a single core.

We believe that, while the experiments here were performed on the Intel Core i5-8279U CPU, many of the findings are likely applicable to all similar Intel processors.

4.1. Test Conditions and Energy Measurement

To evaluate the energy usage (and savings, if any) associated with each of the approaches, we ran, on several volume datasets, each of the gradient estimation and curvature determination methods using different combinations of the strategies previously described. In total, 8 variants of each gradient estimation method were tested (including a **Baseline** variant compiled with **-O2** and using no other energy optimisation strategies) and 70 variants of each curvature determination method were tested (including a **Baseline** variant compiled with **-O2** and using no other energy optimisation approaches). Relative energy use versus the baselines were then found.

To measure energy, PAPI was used to determine total energy usage (in Joules), ζ, of the RAPL PP0 plus the RAPL DRAM domains during just the curvature determination / gradient estimation steps (i.e., excluding I/O associated with loading the data or writing the results). The RAPL PP0 domain represents the power usage of all the processor cores and excluding the DRAM or GPU. The RAPL DRAM domain represents the power usage of the DRAM. This measurement of PP0 + DRAM thus represents the sum of the CPU and DRAM power usage of each method. (N.B., we chose to include the DRAM domain because some of the energy saving strategies, such as **Bespoke**, utilise more memory compared to the baseline, and the inclusion of the DRAM domain ensures that the extra memory usage associated with this is accounted for.)

To ensure consistent results, each variant was run five times and the trimmed means of the PP0 and DRAM domains was computed,

with those two values then summed to give the final energy usage measurement.

4.2. The Datasets and Visualisation

Our experiments consider three types of volumetric data: Marschner-Lobb (**ML**), **Foot**, and **Genus3**. These datasets were used in prior curvature and gradient studies (e.g., [WLMN10, HN20]). ML and Genus3 are synthetic data, while Foot is sensed. ML and Foot are size $256 \times 256 \times 256$ and Genus3 is size $128 \times 128 \times 128$. All volumes were stored as double precision floating point data. ML was considered with and without Gaussian noise ($\sigma = 0.0084$).

A slice image of Foot is shown in Fig. 1a, and a DVR using a curvature-based transfer function is shown in Fig. 1b; Fig. 1b demonstrates one visualisation usage involving the descriptors considered here.

	OP	TE
Baseline	139.95 J	56.71 J
Fast	0.47%	-0.14%
Unroll	-3.56%	-2.88%
Fast, Unroll	-5.64%	-5.42%
O3	-24.27%	-13.17%
O3, Fast	-26.87%	-16.47%
O3, Unroll	-20.81%	-9.85%
O3, Unroll, Fast	-22.98%	-13.61%
Bespoke	-41.54%	-45.02%
Bespoke, Fast	-41.81%	-46.47%
Bespoke, Unroll	-41.68%	-44.47%
Bespoke, Unroll, Fast	-42.33%	-46.36%
Bespoke, O3	-53.25%	-44.87%
Bespoke, O3, Fast	-53.52%	**-47.04%**
Bespoke, O3, Unroll	-52.86%	-44.55%
Bespoke, O3, Unroll, Fast	**-53.72%**	-46.69%
Bespoke, Blocking8, Unroll	-42.08%	-44.71%
Bespoke, Blocking8, Unroll, Fast	-41.98%	-45.64%
Bespoke, Blocking8, O3	-53.13%	-43.74%
Bespoke, Pipeline, O3	-52.53%	-44.37%
Bespoke, Pipeline, O3, Unroll	-52.55%	-43.71%
Bespoke, Pipeline, Blocking8, O3, Unroll	-52.69%	-43.83%
Bespoke, Pipeline, Blocking8, O3, Unroll, Fast	-53.54%	-45.80%

Table 3: *Selected **ML** curvature (energy usage) results.*

4.3. Gradient Estimation

Tables 1 and 2 show gradient energy usage results on **Foot** and **Genus3**, respectively. The first rows in each table present energy usage outcomes for the baseline computations while other rows show energy usage change (as a percentage of Baseline) from use of one of the strategies or combinations of strategies. **Bold** values show the variant with the lowest energy usage. Cell colouring on a red-green scale indicates the degree of improvement. Here, the **Unroll** strategy was quite effective in saving energy. The **O3** strategy, when coupled with **Unroll**, typically achieved modest additional savings. **Fast**, on balance, had a negligible effect.

	OP	TE
Baseline	17.21 J	6.99 J
O3	-24.78%	-12.79%
Bespoke	-49.47%	-48.62%
Bespoke, O3	-55.35%	-48.38%
Bespoke, O3, Fast	-53.63%	**-50.34%**
Bespoke, Blocking8, Unroll	-46.15%	-46.47%
Bespoke, Blocking8, Unroll, Fast	-46.07%	-47.28%
Bespoke, Blocking8, O3	-63.60%	-45.35%
Bespoke, Blocking8, O3, Unroll	-62.64%	-45.57%
Bespoke, Blocking32, O3, Unroll, Fast	**-64.11%**	-48.77%
Bespoke, Blocking64, O3, Unroll, Fast	-63.38%	-49.38%
Bespoke, Pipeline	-50.50%	-43.32%
Bespoke, Pipeline, Blocking8	-45.54%	-43.98%
Bespoke, Pipeline, Unroll	-45.76%	-45.46%
Bespoke, Pipeline, Blocking8, Unroll	-45.50%	-45.70%
Bespoke, Pipeline, Unroll, Fast	-45.87%	-47.21%
Bespoke, Pipeline, Blocking8, Unroll, Fast	-46.02%	-47.89%
Bespoke, Pipeline, O3	-54.01%	-48.12%
Bespoke, Pipeline, Blocking8, O3	-53.92%	-46.54%
Bespoke, Pipeline, O3, Unroll	-56.34%	-47.06%
Bespoke, Pipeline, Blocking8, O3, Unroll	-57.27%	-47.63%
Bespoke, Pipeline, O3, Unroll, Fast	-58.03%	-49.36%
Bespoke, Pipeline, Blocking8, O3, Unroll, Fast	-63.41%	-49.13%

Table 4: *Selected **Genus3** curvature (energy usage) results.*

	OP	TE
Baseline	10.80 sec	4.02 sec
O3	8.89 sec	3.64 sec
Bespoke, Unroll	6.66 sec	2.44 sec
Bespoke, O3, Unroll, Fast	5.34 sec	2.37 sec
Bespoke, Pipeline, Blocking8, O3, Unroll, Fast	5.37 sec	2.41 sec

Table 5: *A selection of **ML** execution times.*

4.4. Curvature Determination Results

Table 3 shows a selection of curvature determination energy usage results on the ML dataset. Its first row presents the energy usage outcomes for the baseline computations for the curvature methods. The table again uses the energy differences and colour coding used earlier. Best results were the combinations: (**Bespoke**, **O3**, **Unroll**, **Fast**) in the case of **OP** and (**Bespoke**, **O3**, **Fast**) in the case of **TE**. These same variants also exhibited the lowest energy usage on the other $256 \times 256 \times 256$ volumes (not shown here).

It is not always the case that the most energy-efficient realisation is the fastest realisation. For the **TE** results in Table 3, for example, the least energy-consuming combination is **Bespoke, O3, Fast**. However, the fastest execution was achieved by the combination of **Bespoke, O3, Unroll, Fast**. Table 5 shows a selection of execution times on the **ML** dataset.

Table 4 shows a selection on energy usage results on the **Genus3** dataset.

For all datasets, use of the **Bespoke** strategy alone produces a large energy savings. Using the **O3** strategy alone also produces a notable energy savings. Combining **Bespoke** with **O3** results in further energy savings, although sometimes only if also combined

	PP0	DRAM
Baseline	136.25 J	3.70 J
O3	102.72 J	3.26 J
Bespoke, O3	59.80 J	5.64 J
Bespoke, Unroll	75.57 J	6.06 J
Bespoke, O3, Unroll, Fast	59.15 J	5.62 J
Bespoke, Pipeline, Blocking8, O3, Unroll, Fast	59.38 J	5.64 J

Table 6: *A selection of **ML** energy uses of **OP**.*

with at least one other strategy. For example, combination of one or, especially, both with **Fast** typically results in additional energy savings. **Unroll** in isolation offers a modest improvement, but it was less effective in combination with other strategies. The **Bespoke** strategy commonly uses a little more energy in its memory-related processing but substantially less energy for other parts of processing, according to reports gathered from RAPL. Table 6 shows selected energy uses (broken out by PP0 and DRAM) on the **ML** dataset. These results reveal that the **Bespoke** strategy very effectively trades off higher memory usage for overall power reduction. For example, in the case of **Bespoke, O3** for **OP** on the **ML** dataset, 2.42 J more DRAM energy is used, but an energy savings of 42.92 J is achieved in PP0 (compared to **O3** alone).

Use of **Blocking** and **Pipeline** often did not produce substantial energy savings. Changes in block size had little impact in the performance of the **Blocking** approach.

4.5. Accuracy Considerations

For some codes, use of certain instruction optimisation settings, in particular mathematics-based optimisation settings, can affect accuracy of results. However, the optimisations used in our strategies appear to not meaningfully degrade accuracy for gradients or curvatures, as discussed next.

For the gradient determination on the **ML** dataset, the most energy-efficient realisation had no difference in gradient values versus the baseline realisation.

For **OP**-based curvature determination on the **ML** dataset, the most energy-efficient realisation had a maximal difference in curvature values of 1×10^{-12} versus the baseline realisation. (That is, no single curvature value from energy-optimal **OP** differed more than a 1×10^{-12} from the corresponding value in the baseline result.) For **TE**'s curvatures for **ML**, the most energy-efficient realisation's values never had a difference (versus the baseline) of more than 4×10^{-12}.

In summary, use of the energy optimisation approaches here can enable achievement of more power-optimal volume data analysis and visualisation apparently without a meaningful impact on accuracy.

4.6. Analyses

The **Bespoke** strategy produced, relative to the other strategies, the most significant energy savings. To determine some of the underlying causes of this occurrence, we explored aspects of its behaviour via Intel's VTune tool. The report and analysis using VTune here is based on a run of the **OP** curvature determination on the **ML** dataset.

In particular, we found that the **Bespoke** strategy was able to utilise the memory access capability of the system more effectively. On this CPU, the maximum memory bandwidth is 20 GB/sec. **OP** using **Bespoke** and **O3** together utilised a maximal bandwidth of 14 GB/sec whereas **OP** using **O3** alone utilised a maximal bandwidth of just 6.7 GB/sec. Additionally, for these scenarios, the average bandwidths utilised, respectively, were 7.96 GB/sec and 1.69 GB/sec.

Additionally, **OP** using **Bespoke** and **O3** together incurred about *a factor of four improvement in* the number of loads and stores incurred by **OP** using **O3** alone. Specifically, **OP** using **Bespoke** and **O3** incurred 4.7×10^9 loads and 1.6×10^9 stores. **OP** with **O3** incurred 17.7×10^9 loads and 6.2×10^9 stores.

Thus, the **Bespoke** strategy has a major benefit of effectively organising and staging the data to allow much more optimal use of the memory channel, decreasing both time and energy consumption.

5. Conclusions and Future Work

In this study of energy usage for the common DVR shading descriptors of gradients and curvatures, a variety of approaches for reducing energy usage were described. The **Bespoke** approach, which aims to decrease energy usage by organising memory accesses associated with convolution in a cache-efficient manner, along with the **Pipeline** and **Blocking** approaches are applicable only to the curvature determination process. Of these, the **Bespoke** approach was among the most successful and was employed by all of the most energy efficient curvature determination variants. The **O3**, **Unroll**, and **Fast** approaches are applicable to both curvature determination and gradient estimation. Of these, the **O3** and **Fast** approaches were among the most successful when applied to curvature determination, with all of the most energy efficient curvature determination variants employing both **O3** and **Fast**. In the case of gradient estimation, nearly all of the most energy efficient variants employed **Unroll**, often in conjunction with **O3**, and none of them employed **Fast**.

Overall, close to 20% energy savings were achieved for gradient descriptor computation, and a factor-of-two improvement in energy efficiency was achieved for curvature descriptor computation. These results are especially significant given that the work was done in C/C++, already an energy-efficient environment [PCR*17]. The findings thus point to energy-efficient descriptor use that can benefit any DVR or analysis environment via calls to energy-efficient routines.

For future work, we hope to consider energy consumption of DVR descriptor computation on GPUs, in particular recent Nvidia GPUs, as recent work has reported [JOL*23] that the NVidia Management Library (NVML) API reports power consumption within 5% accuracy on the Fermi-class and newer GPUs. (For earlier GPUs, such as Tesla-class GPUs, challenges in achieving accuracy using the GPU's built-in power sensor have been reported [BZZ14].)

References

[ACL17] ALHARBI N., CHAVENT M., LARAMEE R. S.: Real-Time Rendering of Molecular Dynamics Simulation Data: A Tutorial. In *Computer Graphics and Visual Computing (CGVC)* (2017), Wan T. R., Vidal F., (Eds.), The Eurographics Association. doi:10.2312/cgvc.20171277. 1

[BD21] BROWNLEE C., DEMARLE D.: Chap. 45, fast volumetric gradient shading approximations for scientific ray tracing. In *Ray Tracing Gems II* (2021), Marrs A., Shirley P., Wald I., (Eds.), Apress. 1

[BDG*00] BROWNE S., DONGARRA J., GARNER N., HO G., MUCCI P.: A portable programming interface for performance evaluation on modern processors. *Int'l J. High Perf. Comp. Appl. 14*, 3 (2000), 189–204. 2

[BZZ14] BURTSCHER M., ZECENA I., ZONG Z.: Measuring gpu power with the k20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs* (New York, NY, USA, 2014), GPGPU-7, Association for Computing Machinery, p. 28–36. doi:10.1145/2588768.2576783. 6

[Fre23] FREE SOFTWARE FOUNDATION: Options that control optimization. URL: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. 3

[FZZ*21] FALUDI B., ZENTAI N., ZELECHOWSKI M., ZAM A., RAUTER G., GRIESSEN M., CATTIN P. C.: Transfer-function-independent acceleration structure for volume rendering in virtual reality. In *Proc., Conf. on High-Performance Graphics* (2021). 1

[HHR*20] HENDERSON P., HU J., ROMOFF J., BRUNSKILL E., JURAFSKY D., PINEAU J.: Towards the systematic reporting of the energy and carbon footprints of machine learning. *J. Machine Learn. Res. 21* (2020), 1–43. 2

[HN20] HAUENSTEIN J. D., NEWMAN T. S.: Descriptions and evaluations of methods for determining surface curvature in volumetric data. *Computers & Graphics 86* (2020), 52–70. 2, 5

[HOK*16] HIRKI M., OU Z., KHAN K. N., NURMINEN J. K., NIEMI T.: Empirical study of power consumption of x86-64 instruction decoder. In *Proc., USENIX Cool Topics in Sustainable Data Centers* (2016), USENIX, pp. 1–6. 2

[JOL*23] JAY M., OSTAPENCO V., LEFÈVRE L., TRYSTRAM D., ORGERIE A.-C., FICHEL B.: An experimental comparison of software-based power meters: focus on cpu and gpu. In *CCGrid 2023-23rd IEEE/ACM international symposium on cluster, cloud and internet computing* (2023), IEEE, pp. 1–13. 2, 6

[KWTM03] KINDLMANN G., WHITAKER R., TASDIZEN T., MOLLER T.: Curvature-based transfer functions for direct volume rendering: methods and applications. In *Proc., IEEE Visualization, 2003* (2003), pp. 513–520. doi:10.1109/VISUAL.2003.1250414. 1, 2

[LV23] LAPLANTE P., VOAS J.: "Frameworking" carbon-aware computing research. *IEEE Computer 56* (2023), 105–108. 2

[PCR*17] PEREIRA R., COUTO M., RIBEIRO F., RUA R., CUNHA J., FERNANDES J., SARAIVA J.: Energy efficiency across programming languages. In *Proc., ACM SIGPLAN Int'l Conf. Soft. Lang. Engg.* (2017), IEEE, pp. 256–267. 2, 6

[RPSC99] RAY H., PFISTER H., SILVER D., COOK T.: Ray casting architectures for volume visualization. *IEEE Trans. on Vis. and Computer Graphics 5*, 3 (1999), 210–223. 1

[SLYP22] SUN J., LENZ D., YU H., PETERKA T.: Mfa-dvr: Direct volume rendering of mfa models. *Arxiv:2204.11762* (2022). 1

[ST03] SENG J. S., TULLSEN D. M.: The effect of compiler optimizations on pentium 4 power consumption. In *Proc., Seventh Works. on Interaction Between Compilers and Comp. Architectures* (2003), IEEE, pp. 51–56. 2

[TBF*22] TARNER H., BRUDER V., FREY S., ERTL T., BECK F.: Visually comparing rendering performance from multiple perspectives. In *Proc., Vision, Modeling, and Visualization '22* (2022), pp. 115–125. 1

[TWD*13] THIYAGALINGAM J., WALTON S., DUFFY B., TREFETHEN A., CHEN M.: Complexity plots. *Computer Graphics Forum 32*, 3pt1 (2013), 111–120. doi:https://doi.org/10.1111/cgf.12098. 2

[Vas15] VASILAKIS E.: An instruction level energy characterization of arm processors. *Foundation of Res. and Technology Hellas, Inst. of Comput. Science, Tech. Rep. FORTH-ICS/TR-450* (2015). 2, 3

[WJK*12] WEAVER V. M., JOHNSON M., KASICHAYANULA K., RALPH J., LUSZCZEK P., TERPSTRA D., MOORE S.: Measuring energy and power with papi. In *41st Int'l Conf. Par. Proc., Works.* (2012), IEEE, pp. 262–268. 2

[WLMN10] WOOD B. A., LEE J. K., MASKEY M., NEWMAN T. S.: Higher order approximating normals and their impact on isosurface shading accuracy. *Machine Graphics & Vision 19*, 2 (2010), 201–221. 2, 5

[WRN05] WEIN W., ROEPER B., NAVAB N.: 2d/3d registration based on volume gradients. In *Proc., SPIE 5747 Medical Imaging '05: Image Processing* (2005), pp. 144–150. 1

[XTC*22] XU J., THEVENON G., CHABAT T., MCCORMICK M., LI F., BIRDSONG T., MARTIN K., LEE Y., AYLWAR S.: Interactive, in-browser cinematic volume rendering of medical images. *Comp. Meth. Biomech. and Biomed. Engg.: Imaging & Vis.* (2022). 1

[ZSRH19] ZORRILLA F., SAPPL J., RAUCH W., HARDERS M.: Accelerating Surface Tension Calculation in SPH via Particle Classification and Monte Carlo Integration. In *Computer Graphics and Visual Computing (CGVC)* (2019), Vidal F. P., Tam G. K. L., Roberts J. C., (Eds.), The Eurographics Association. doi:10.2312/cgvc.20191260. 1