On the energy complexity of Algorithms realized in CMOS, a Graphics Example

J. Smit, M. Bosma

University of Twente
Department of Electrical Engineering EF9250
POBox 217, 7500AE Enschede, The Netherlands
e-mail: jaap@nt.el.utwente.nl,
bosma@nt.el.utwente.nl

Abstract

A theory about the energy consumption of *algorithms* realized in CMOS, presented in related work, makes it possible to calculate the minimal amount of energy dissipated for the execution of an algorithm. The rendering of a dense dataset with three variants of the Volume Rendering algorithm will be considered as an example of the methodology. The absolute lower bound of the energy consumption is calculated for the rendering of a dense 256^3 dataset using implementations of the algorithms in an 1µm CMOS process. Predictions of the energy consumption in future CMOS generations are given as well.

1 Background

Computer graphics applications are so computationally intensive that it is considered to be worth while to consider the construction of a dedicated Volume Rendering Chip [1], [2], [3], [4], or specific DSP processors. The overall speed required for volume rendering is roughly a factor 512 larger than the speed required for a 2D graphics application. The energy consumption of the algorithm is hence of major importance for the feasibility of a real-time renderer which could be used in a notebook. Design alternatives can be compared with some ease, as soon as the energy complexity of the algorithm considered is known. Let us first look at the energy consumption for a state of the art RISC-CPU which renders a *dense* 256^3 dataset using a 100MHz processor, dissipating 5W. It takes 120 seconds of CPU-time to generate a single image. The amount of energy consumed is hence 120 x 5 = 600 Joule. The use of multiple CPU's to solve the task at a view-rate of 25 images per second, implies a power consumption of 25 x 600 = 15 kWatt. It will be shown that the power consumption of this algorithm has a lower bound of a mere 52 Watt in a 1µm CMOS process. I.e. the amount of energy dissipated by the RISC CPU is a factor of 288 higher than really needed. The RISC-CPU offers however flexibility, which a totally hard wired CMOS chip would not have. This makes it for instance possible to render a sparse dataset in a few seconds. This reduces the power consumption to a few hundred Watt. Similar savings are however possible for dedicated datapaths, which use considerably less energy. The calculation of the energy complexity does not only result in a number for the lower bound for the energy consumption of the algorithm, it includes as well some global rules about the design, governing the *placement* and *routing* of blocks, the *partitioning* of the algorithm, the uses of memories to store and retrieve (intermediate) results, etc.

The approach taken in this paper is complementary to the commonly used *synthesis*, *design* and *analysis* approach, which is well described in [6] and [7] for low power applications, as it introduces an extra analysis layer, in which the fundamental properties of the underlying *algorithm* are made explicit in terms of a calibrated technology reference, which describes the fundamental properties of the underlying CMOS process. The technology reference is not calibrated in terms of a library, in order to make the complexity figures library independent.

2 Introduction to the problem

It will be shown how the minimal amount of energy, needed to compute a dense super-resolution [8], [9], rendering of a 256^3 dataset viewed with an oversampling of a factor of two in all three directions at a rate of 25 images per second, can be derived from three very basic quantities which characterize fundamental properties of the underlying CMOS process, which are: 1) the energy consumption of a single bit-addition performed by a full-adder in the context of an addition and a multiplication, 2) the energy needed to transfer a signal value over a wire of unit-length, and 3) the size of a storage-cell in a random access memory.

It is shown in the paper in which these concepts were introduced [11], that these three properties characterize the *arithmetic*, *communication* and *datastorage* aspects of the underlying CMOS process in which the algorithm should be implemented. The idea of energy complexity can only be solved when certain aspects of *layout*, *placement* and *partitioning* of a VLSI design are known. The complicated nature of these concepts has resulted in a design-practice in which the problem is transformed in a logic description, which is fed into a synthesis tool in the hope that the energy consumption will stay within bounds. It will be shown in the next section that the aspects of placement, routing and

partitioning need not be solved in full detail to be able to predict the energy consumption within bounds. I.e. it will be shown that constraints can be formulated which imply *design guidelines* which can be imposed on the (use of the) tools in the design process.

3 Methodology

A short description of the methodology introduced in [11] is needed to give the reader a proper understanding of the underlying theory, its inherent precision as well as the simplifications made which make it possible to derive a lower bound for the energy complexity of an algorithm.

The *arithmetic energy complexity* of an arithmetic function or a combinatorial algorithm can be derived from the minimal average energy $\mathcal{E}_{fa}$, needed to perform a bit-addition. This quantity is evaluated through the careful design of a full-adder and its subsequent characterization in the target CMOS process (see for instance [6] p. 91, [10] & [11]). This results in a lower bound of $\mathcal{E}_{fa} = 2.41$ pJ for a standard 1µm, 5V CMOS process. The energy needed for *on-chip communication* is derived from the energy needed to switch 1m of (metal1) wire. This amounts to: $\mathcal{E}_{/m} = 1.44$ nJ in the same process. The *energy needed for datastorage* can be calculated, using an appropriate RAM-model, from the diameter of a RAM-cell $\mathcal{D}_{cell}$, which is by definition the sum of the width $\mathcal{W}_{cell}$ and the height $\mathcal{H}_{cell}$ of the RAM-cell. $\mathcal{D}_{cell} = 40$µm for an on-chip SRAM in the 1µm 5V CMOS process.

Predictions of the overall power consumption for the algorithm for future CMOS generations will be given later on.

The aspect of *energy optimal placement* of arithmetic building blocks becomes operational through the introduction of the *power radius* concept. The value of the power radius $\mathcal{R}_{pd}$ is defined as the length of the wires connected to a cell, such that the energy needed to transfer data over the wires equals the average energy needed to perform the basic operation of the cell.

The *RAM-model* makes it possible to find an optimal size for the memories involved, given specific access patters for an *optimal partitioning* of the *algorithm* at hand.

4 A second level of abstraction

The value of the power radius of a full-adder is easily calculated from the figures given:

$$\mathcal{R}_{pd\_fa} = \mathcal{E}_{fa} / (P_t \; n \; \mathcal{E}_{/m}) = 2.41\;10^{-12} / (0.5\;5\;1.44\;10^{-9}) = 0.67 \text{ mm}$$

With P_t the chance on a signal transition and n the number of wires connected to the full-adder. The rather small value of $\mathcal{R}_{pd\_fa}$ makes the placement problem quite complicated and important. A way to make things better accessible is to work with larger arithmetic units, like adders and multipliers. There is however a problem to be solved. The actual construction of adders and multipliers from full-adders results in unnecessary signal transitions which cause excess energy consumption. This phenomenon is modeled in [11] by a ripple factor ϱ which is a measure for the excess energy is needed. The actual value of ϱ will depends on the technique used to perform the operation, be it a fully combinatorial or asynchronous. The tendency is that simple combinatorial techniques dissipate least energy, whereas circuits with carry look-ahead or carry select circuitry, consume progressively more energy to perform one single operation.

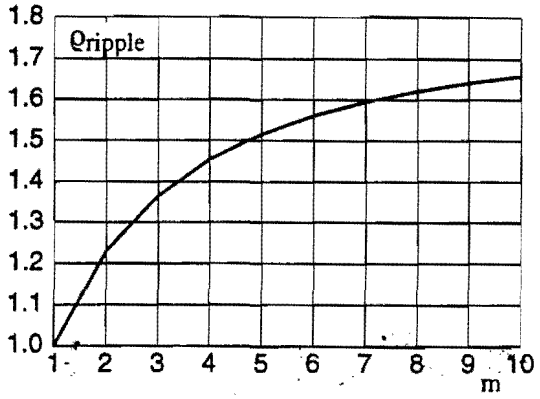

Fig. 1: The ripple factor ϱ_{ripple}

Simple formulas can now be used to describe the minimal amount of energy at the next higher level of hierarchy. For instance an m-bits wide ripple carry adder will dissipate an amount of energy:

$$\mathcal{E}_{ripple} = m \; \varrho_{ripple} \; \mathcal{E}_{fa}$$

The appropriate value for the ripple factor can be taken from the graph in Figure 1.

The energy consumption $\mathcal{E}_{\_rc}$ of an m-bits wide ripple carry adder is hence for m=8:

$$\mathcal{E}_{\_rc} = m \; \varrho_{ripple} \; \mathcal{E}_{fa} = 8\;1.64\;2.41\;10^{-12} = 31.6 \text{ pJ}$$

The value of $\mathcal{R}_{pd}$ is for m=8:

$$\mathcal{R}_{pd\_rc} = m \; \varrho_{ripple} \; \mathcal{E}_{fa} / (P_t\;3\;m\;\mathcal{E}_{/m}) = 31.6\;10^{-12} / (0.5\;3\;8\;1.44\;10^{-9}) = 1.8 \text{ mm}$$

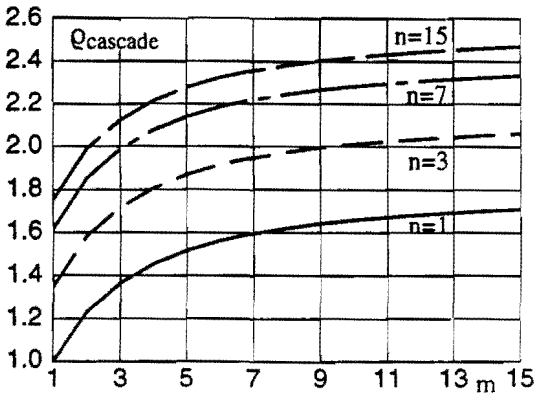

Fig. 2: $\varrho_{cascade}$

Similar results were derived for multipliers in [11]. Figure 2 shows the ripple factor of a cascade adder structure used in an array multiplier.

The energy consumption $\mathcal{E}_{mul}$ of an mxn array multiplier is for m=n=8:

$$\mathcal{E}_{mul} = m\ n\ (\varrho_{cascade}\ \mathcal{E}_{fa} + \mathcal{E}_{and}) = 8\ 8\ (2.3\ 2.41\ 10^{-12} + 0.5\ 10^{-12}) = 387\ pJ$$

The value of $\mathcal{R}_{pd}$ is for m=n=8:

$$\mathcal{R}_{pd\_mul} = \mathcal{E}_{mul} / (P_t\ 4\ m\ \mathcal{E}_{/m}) = 31.6\ 10^{-12} / (0.5\ 4\ 8\ 1.44\ 10^{-9}) = 16.8\ mm$$

The power radius, $\mathcal{R}_{pd}$, is 100x the width of the multiplier and 24x its height. This large value of the power radius of the multiplier makes it relatively simple to *place and route* the functional blocks, used in the algorithms considered, in such a way that just a fraction of $\mathcal{R}_{pd}$ is used for wiring, provided that all adders are placed adjacent to the multiplier with which they share connections.

5 Energy consumption of on-chip RAMs

The length of the wires used to transfer bits in a memory from the 'consumer' location to the destination address plays an important role in the estimation of the energy consumption of a random access memory (RAM).

Figure 3 shows a sketch of a RAM, in which the width and height of the RAM-cell are shown as $\mathcal{W}_{cell}$ and $\mathcal{H}_{cell}$. The diameter of the RAM-cell is by definition:

$$\mathcal{D}_{cell} = \mathcal{W}_{cell} + \mathcal{H}_{cell}$$

For a square RAM, we have a diameter of the RAM which equals: $\mathcal{D}_{RAM} = \sqrt{N}\ \mathcal{D}_{cell}$. The energy consumption of an ideal RAM is, for a signal transition probability P_t on the data-lines:

$$\mathcal{E}_{ideal} = \mathcal{D}_{RAM}\ \mathcal{W}\ P_t\ \mathcal{E}_{/m}$$

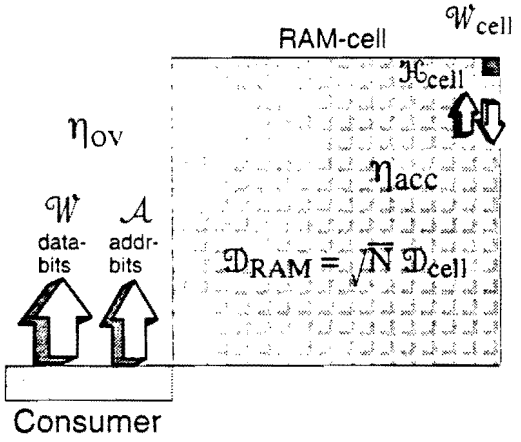

Fig. 3: A RAM architecture

It should be noted that not only $\mathcal{W}$ databits have to be transferred to the cell position, an amount of $\mathcal{A}$ address bits is needed as well to select the cell. This fact is encapsulated in the overhead efficiency factor $\eta_{ov} = \mathcal{W} / (\mathcal{W} + \mathcal{A} + 2)$. The term 2 in the denominator models the fact that the row-select line makes two transitions in one datatransfer cycle.

It was shown in [11] that the actual access to RAM-cells can be made, using an hierarchical RAM-structure, with an efficiency η_{acc} which approaches 1.

Using the efficiency parameters just introduced we have:

$$\mathcal{E}_{RAM} = \mathcal{D}_{RAM}\ \mathcal{W}\ P_t\ \mathcal{E}_{/m} / (\eta_{ov}\ \eta_{acc})$$

Most on-chip SRAMs as well as the larger external RAMs have a value of η_{acc} which is in the order of 1/8. This more realistic value, which can be implemented with some care, will be used in the actual energy calculations for all RAMs used in the energy calculations for the volume rendering architectures proposed.

It should be noted that a square memory outline is assumed. This assumption can be made valid for *on-chip* RAMs. It is not realistic to construct a volume renderer with on-chip RAM only. Hence it is necessary to consider *off-chip* RAMs as well for the calculation of the energy consumption of a complete system.

The RAM-core within a modern memory chip is generally not square. Moreover, there is a tendency to implement memory-access as a two-step (RAS-CAS) process. This makes off-chip RAMs frequently less efficient than their on-chip counterparts. It should be noted however that the energy complexity of off-chips RAMs is of a completely different nature. The fact that multiple chips are used makes it possible to implement access to memories with a size which is just a fraction of the total amount, at the cost of additional communication over IO-pins. What this means for practical off-chip RAMs will be analyzed in the next section.

6 External random access memories

Figure 4 shows a memory organization in which a RAM-core is read and written over a width of 512 bits into a sequential-access memory.

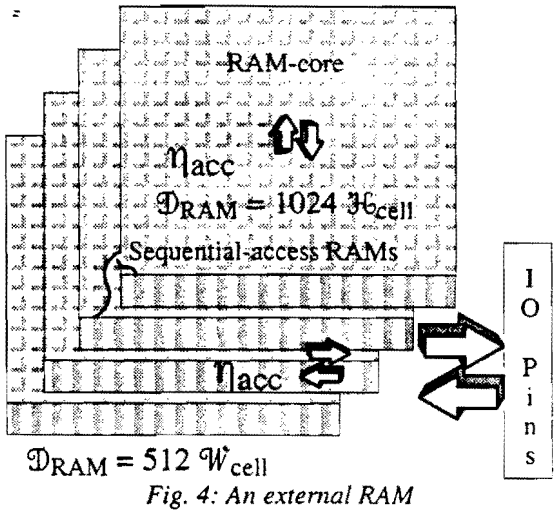

Fig. 4: An external RAM

Such memories have the ability to transfer data very fast when special purpose IO-circuits are used. The IO-pins

terminate the on-board wiring in such a way that an ideal transmission-line with characteristic impedance $Z_0 = \sqrt{l/c}$ is formed. A relatively low voltage swing is typically used to transfer the data.

Adaptation of the methodology of the previous section to this non-square and hence non-optimal, two-stage RAM structure gives, for a total of $\mathcal{W} = 8$ core-rams,

$$\mathcal{E}_{core} = \mathcal{W}\ 512\ P_t\ \mathcal{E}_{/m}\ \mathcal{H}_{cell}\ 1024 / (\eta_{ov}\ \eta_{acc})$$

with:

$$\eta_{ov} = 512 / (512 + 10 + 2)$$

$$\eta_{acc} \approx 1/8$$

and for the sequentially accessed border RAMs:

$$\mathcal{E}_{border} = \mathcal{W}\ P_t\ \mathcal{E}_{/m}\ \mathcal{W}_{cell}\ 512 / (\eta_{ov}\ \eta_{acc})$$

with:

$$\eta_{ov} = 8 / (8 + 5 + 2)$$

$$\eta_{acc} \approx 1/8$$

The IO-circuits shown cause dissipation of energy as well. A 16 MByte memory can be constructed from 32 0.5 MByte chips. The capacitance of IO-pins is typically in the order of 5 pF. In a transmission-line IO-organization these capacitances decrease the value of the characteristic impedance $Z_0 = \sqrt{l/c}$ with l and c the inductance and capacitance of the transmission-line per unit length. The value for Z_0 is, with the extra capacitance of the IO-pins included, typically 100 Ω. A transmission-line structure is to be terminated with 100 Ω resistors on both ends. Hence there will be an impedance of 50 Ω at a point at which data is inserted somewhere between the terminations of the transmission-line. The amount of energy, dissipated by a driver, fed from a power supply V_p, transmitting a byte-burst of S cycles, and $\mathcal{A}_s$ address cycles, over a transmission-line using a voltage swing Vs on the line at a transfer-rate $1 / T_b$ is:

$$\mathcal{E}_{Z0} = (S + \mathcal{A}_s)\ T_b\ (\mathcal{W} + 1)\ V_p\ V_s\ 2 / Z_0$$

Where $\mathcal{W}$ is the word-width, i.e. $\mathcal{W} = 8$. Note that the sequential access is assumed to be clocked on both the positive and the negative clock-edge, this is why the effective width of the data transfers is taken to be $(\mathcal{W} + 1)$.

Practical values are:

$$V_p = 5V,\ V_s = 0.5\ V,\ Z_0 = 100\ \Omega,\ \mathcal{A}_s = 2$$

It is as well possible to use the traditional capacitive IO-circuitry in conjunction with VRAMs. The energy consumption for a byte-burst is in this case:

$$\mathcal{E}_{Cap} = (S + \mathcal{A}_s)\ (\mathcal{W} + 1)\ N\ P_t\ C_{in}\ V_p^2$$

Practical values are:

$$V_p = 5V \text{ or } 3V,\ \mathcal{A}_s = 2,\ N = 32,\ P_t = 0.5,\ C_{in} = 5pF$$

Figure 5 shows the amount of energy needed to transfer a byte-burst over the IO-Pins of the RAM-chip, as well as the amount of energy $\mathcal{E}_{data}$, needed to transfer the byte-burst.

The figure shows clearly that capacitive IO-circuits may dissipate as much energy as is needed for retrieval of the voxel-data from the RAM-chips, whereas transmission-line configurations dissipate negligible extra energy for the actual data transfer. It can be seen as well that the amount of energy for a burst data transfer is constant over a wide range of burst lengths. This is due to the fact that most energy is dissipated in the RAM-core, which reads the data of a complete column, no matter what the length of the burst is. This is a major disadvantage of most commercially available RAM structures.

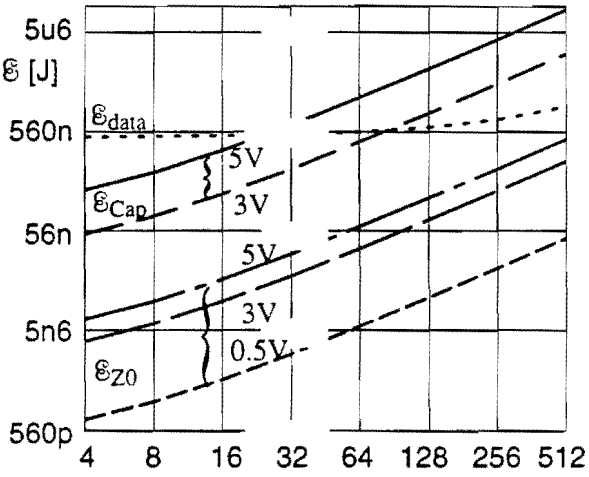

Fig. 5: Energy needed for a byte-burst

7 The volume rendering algorithm

The volume rendering algorithm traces all rays, defined fo (i, j) pairs on a screen S (i, j), at all screen depths k, at ar oversampling rate 2 in all three dimensions over a scala dataset of 256^3 points defined on the 3D grid V wit grey-values $g(\mathbf{x}) \mid \mathbf{x} \in V$. The function RESAMPLE uses a interpolation filter to compute the grey-value: grey_val a the sample location $\mathbf{x}_{i,j,k}$ which coincides with a gri position i,j,k of the screen-space. A non-linear function OP computes the opacity value α from the grey-value at th sample location $\mathbf{x}_{i,j,k}$. The function NABLA computes th gradient vector of the grey value dataset g at the sampl location $\mathbf{x}_{i,j,k}$. A tabular approach is used to compute th color C, from the gradient and the direction of the ligh vector. The opacity and color along the ray, (α_r, C_r) ar

computed in front to back order, with the composition function COMPO.

```
FOREACH ray IN S (i,j) DO
    FOREACH x_{i,j,k} FROM x0
        STEP Δx IN V DO
            grey_val= RESAMPLE (g, x_{i,j,k}),
            α = OPA (grey_val),
            gradient = NABLA (g, x_{i,j,k}),
            C  = TABLE (gradient),
            (α_r,C_r) = COMPO (α,C,α_r,C_r)
    ENDFOR,
ENDFOR:
```

The direct implementation of the volume rendering algorithm shown, using a:

1 Tri-linear resample function in combination with a gradient algorithm introduced in [8], called the intermediate difference gradient, or
2 An approximating spline resample function [13] in combination with the same gradient algorithm, or
3 A cubic-spline resample function in combination with a gradient filter based on the derivative of the cubic-spline function, described in [12],

is in all cases inferior as the amount of energy needed to access the main-memory exceeds the amount of energy needed for actual arithmetic by two to three orders of magnitude. Hence the main objective of the remainder of this paper will be to reformulate, c.q. *repartition*, the algorithm in such a way that the energy consumption due to data transfers becomes negligible compared to the energy consumption due to arithmetic. It will be stated without explicit proof that the amount of energy needed for the RESAMPLE and the NABLA operators is large with respect to the energy needed to execute all remaining operators.

Figure 6 gives a global partitioning of the algorithm, based on object space subdivision, showing the use of a global main memory, an 8x8x8 v-cache used to store voxels on-chip, the main arithmetic section, and the composition section. A disadvantage of the object space partitioning is the need to save and restore intermediate opacity and color values (α_r,C_r) along the rays traversed in a bundle memory. Object space subdivision introduces overhead, as shown in figure 7, due to the necessity to provide for overlap between subvolumes.

The overhead factor is shown as a function of the operator size and the subvolume size in Figure 7. The net amount of energy needed to read a 256^3 dataset once from a main memory over a transmission-line interface with 64 byte bursts is:

$$\mathcal{E}_V = 256^3\ \mathcal{E}_{data} / 64 = 0.147\ \text{J}.$$

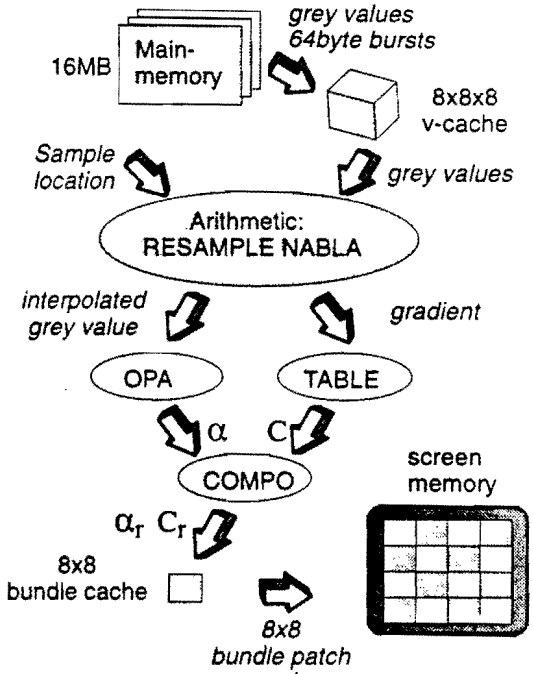

Fig. 6: Object space subdivision

The rather flat curves in Figure 5 indicate that it is quite well possible to decrease the energy consumption of the main memory with a factor close to 4 when the rendering is ordered in such a way that a full subvolume is rendered in an inner loop which is constrained to a single RAS-address range.

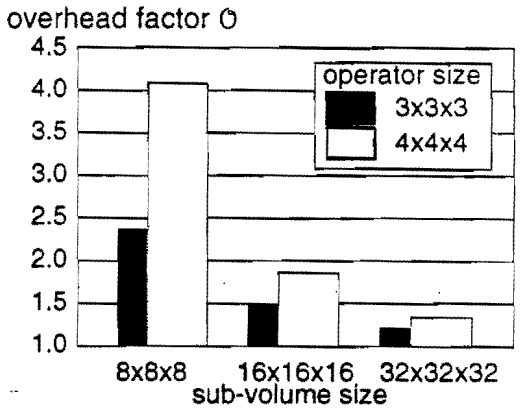

Fig. 7: Overhead

8 Resampling and gradient operators

The actual resampling and gradient algorithms will not be described here, instead just their flowgraph and characteristic figures will be presented. All algorithms contain a large amount of multipliers and adders. It is important to place adders and subtracters close to the multipliers and to use a small fraction of the power radius for the wiring between the multipliers. This cannot be a big problem, as the value of $\mathcal{R}_{pd}$ is 100x the width of the multipliers and 24x its height. The access to memories in

which the place variant coefficients of spline-functions are stored will be neglected for reasons of simplicity. Pipeline registers introduced for speed, or controllers introduced for reasons of flexibility and/or programmability of the data path do, *by definition*, not contribute to the energy complexity of the algorithm, as they are not characteristic for the algorithm and its execution.

9 Arithmetic complexity: Algorithm 1

A signal-flow graph for the calculation of the tri-linear interpolated grey-value grey_val and its gradient $\delta g/\delta x1$, $\delta g/\delta x2$, $\delta g/\delta x3$ is given in Figure 8.

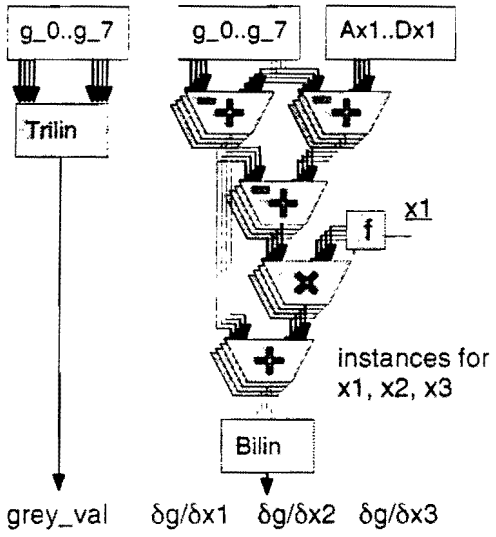

Fig. 8: Flow graph of algorithm 1

A total amount of 28 multiplications and 80 additions /subtractions is used in the flowgraph of Figure 8. The amount of energy consumed by the circuit shown is:

$$\begin{aligned}\mathcal{E}_{ar/tri\text{-}lin} &= 52\ \mathcal{E}_{ripple} + \\ &\quad 28\ [\mathcal{E}_{mul} + \mathcal{E}_{add}] \\ &= 52\ m\ \varrho_{ripple}\ \mathcal{E}_{fa} + \\ &\quad 28\ m\ [n\ (\varrho_{mul}\ \mathcal{E}_{fa} + \mathcal{E}_{and}) + \varrho_{mul}\ \mathcal{E}_{fa}]\end{aligned}$$

This gives for m = n = 8 an energy consumption of 12.2 nJ for each sample location or 1.64 J for each full view of the dataset, i.e. for all 512^3 sample locations.

10 Arithmetic complexity: Algorithm 2

The tri-point spline resampling function is an *approximation* function instead of an *interpolation* function. A total of 2 multiplications are needed for a single interpolation step. A full 3D tri-point interpolation uses 19 additional multiplications and 38 additional additions. The differentiation algorithm is the same as the one used for algorithm 1.

This gives:

$$\begin{aligned}\mathcal{E}_{ar/tri\text{-}point} &= 71\ \mathcal{E}_{ripple} + \\ &\quad 47\ [\mathcal{E}_{mul} + \mathcal{E}_{add}] \\ &= 71\ m\ \varrho_{ripple}\ \mathcal{E}_{fa} + \\ &\quad 47\ m\ [n\ (\varrho_{mul}\ \mathcal{E}_{fa} + \mathcal{E}_{and}) + \varrho_{mul}\ \mathcal{E}_{fa}]\end{aligned}$$

This gives for m = n = 8 an energy consumption of 22.5 nJ for each sample location or 3.02 J for each full view of the dataset.

11 Arithmetic complexity: Algorithm 3

The cubic-spline resampling function is either an interpolation function or an approximation function depending on the parameters selected. A full 3D cubic-spline interpolation uses 16 x 4 + 4 x 4 + 1 x 4 = 84 multiplications and 16 x 3 + 4 x 3 + 1 x 3 = 63 additions. The differentiation algorithm uses the same algorithm with a different spline-table content for each component of the gradient This gives:

$$\begin{aligned}\mathcal{E}_{ar/cubic\text{-}spline} &= 141\ [\mathcal{E}_{mul} + \mathcal{E}_{add}] + 47\ \mathcal{E}_{mul} \\ &= 47\ m\ n\ \varrho_{mul}\ \mathcal{E}_{fa} + \\ &\quad 141\ m\ [n\ (\varrho_{mul}\ \mathcal{E}_{fa} + \mathcal{E}_{and}) + \\ &\quad \varrho_{mul}\ \mathcal{E}_{fa}]\end{aligned}$$

This gives for m = n = 8 an energy consumption of 77.4 nJ for each sample location or 10.4 J for each full view of the dataset.

12 The global architectural model

The presence of an external main memory, backed up by a (double buffered) 8x8x8 voxel-cache, which is partitioned over 8 smaller v-cache units, will be assumed. The arithmetic units use these values to produce the interpolated value as well as the gradient at the sample location. The shader, whose energy complexity is less than 1/9 of the lowest complexity algorithm, produces the color from the gradient, using an 1.5 KByte on-chip table [9] to fetch four color values all at once. A bilinear interpolation is used to compute the proper color from the values stored in the table.

13 Voxel-cache read operations

The amount of energy needed for one access to an 8x8x8 v-cache constructed from 8 sub-caches of size 8x8, is:

$$\mathcal{E}_{cache\_cycle} = \mathcal{D}_{cache}\ \mathcal{W}\ P_t\ \mathcal{E}_{/m}\ /(\eta_{ov}\ \eta_{acc})$$

Using:

$\mathcal{D}_{cache} = \sqrt{8x8}\ \mathcal{D}_{cell}$;

$\mathcal{W} = 8$; $P_t = 0.5$; $\eta_{acc} = 1/8$;

$\eta_{ov} = \mathcal{W} / (\mathcal{W} + \log2\ (8 \times 8) + 2)$;

This gives for each of the eight sub-caches.

$\mathcal{E}_{cache\_cycle} = 709$ pJ

The number of read-cycles for a complete view of all 512^3 sample locations is given in table 1, together with the amoun

of energy needed to read the v-cache, for each of the algorithms.

	read-cycles	$\mathcal{E}_v$ [J]
tri-lin	20×512^3	0.238
tri-point	27×512^3	0.321
cubic-spline	64×512^3	0.761

Table 1

14 Bundle memory read and write

The opacity and color on a ray: (α_r, C_r) should be read from the screen-memory when a certain part of the object space is to be rendered, and be written back when the rendering is ready. This can be done for each ray separately, but it can be organized as well in such a way that groups of rays are covered by, say an 8x8 patch of the screen. The intermediate opacity and color (α_r, C_r) have a double precision accuracy, so an 8x8 patch can be organized as an 128 byte burst. About 15 bursts are at most needed to cover one sub cube at a super sampling rate of 2. as illustrated in Figure 9.

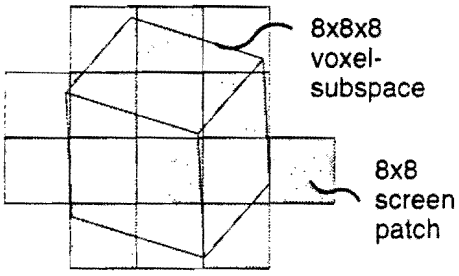

Fig. 9:Bundle memory patches

The use of an off-chip memory with a transmission-line interface results in an energy consumption of 3 mJ for the update of (α_r, C_r) along the rays. This is a negligible amount, compared to all other contributions to the energy complexity.

15 Survey of results

Figure 10 shows the major contributions to the energy complexity. It is clearly seen that the minimal amount of energy for arithmetic is the major contribution to the energy complexity of the volume rendering algorithm. The amount of energy needed for wiring is negligible, as just a small fraction of the power radius of the multiplier needs to be used for wiring, provided that all adders/subtracters are placed immediately adjacent to the multipliers. The amount of energy needed for memory access gives, after the arithmetic energy, the most significant contribution to the energy complexity.

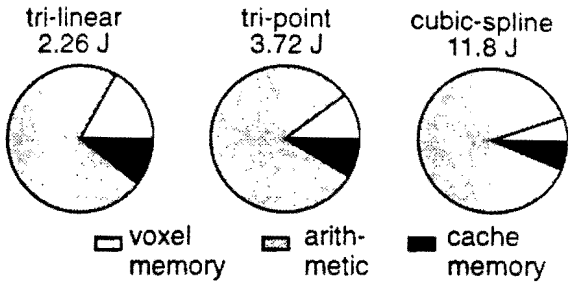

Fig. 10: Energy complexity of the algorithms

16 Possible improvements

Provided that it is impossible to improve the technological basis, one cannot improve the arithmetic energy complexity of the volume rendering algorithm. The arithmetic contribution however is responsible for over 73% of the energy consumption. The assumption of an access efficiency of 1/8 for all memories introduced indicates that memory systems which consume less energy might be possible. The assumption of an 100% access efficiency would however be unrealistic. Nevertheless, there seems room for improved η_{acc} values which come close to 1/2 instead of the value 1/8 which has been used in all formulas. Inclusion of address transition detection hardware in the v-cache reduces the energy consumption of the v-cache to 2/3 of its current value.

It should be well understood that the partitioning of the v-cache into 8 coherently operated v-caches has as effect that the energy consumption for cache read operations is decreased with a factor $\sqrt{8}$. The cache write operations are faced with an hierarchical memory of the original size. One may consider it as a likely realization option to merge the interpolation circuits with all 8 v-cache memories. This has as net effect that v-cache write operations have to drive longer wires than v-cache read operations. This is not a significant problem as the amount of read operations is a factor 20, 27 or even 64 larger, depending on the algorithm selected, than the amount of write operations.

17 Technology outlook

The learning curve of semiconductor foundries has shown a growth of the size of RAMs with a factor of about 2 each 1.5 years, over the last decade.

Figure 11 shows the global effect of this growth on the actual amount of power consumed, assuming that all parts of the algorithm scale equal, for a view rate of 25 images. Another important technology change currently being considered is the use of lower V_{DD} voltages. A decrease of V_{DD} from the 5V to 2.5V gives yet another decrease of a factor of 4 for the power consumption.

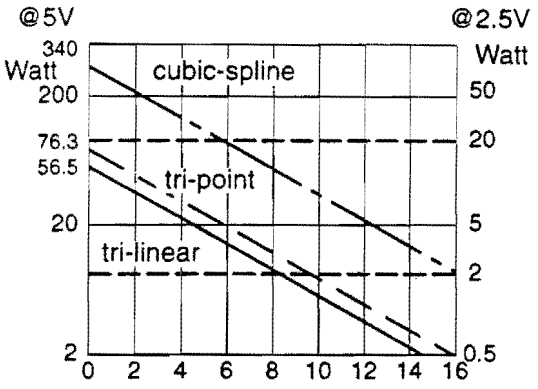

Fig. 11: Technology outlook

Under the assumption that the learning curve of the industry does not diminish, it can be shown that it will be impossible to create a practical notebook which dissipates less than 2W for a fully dense rendering of a 256^3 dataset within 8 years from the introduction of the 1μm process. It will take even longer when the learning curve gets saturated.

18 Single chip sparse rendering

It should be noted that it is not needed to restrict the global architecture described to the rendering of dense datasets. The algorithmic benchmarking program Vivid [13] has been used to study the effect of specific measures which can be implemented either to speed up a software implementation of the volume rendering problem, or to find the major aspects to be implemented in dedicated volume rendering engine. This has shown that a speed improvement of a factor of 120 is possible when large areas are opaque such that the O (N^3) time and energy complexity is changed in an O (N^2) complexity. This results in a dramatic reduction of the energy complexity, which can be realized with relatively little overhead. The actual energy consumption depends now on the actual sparsity of the dataset and the opacity settings used. The two huge factors of 288 for a dedicated design and 120 for a typical sparse problem makes real-time volume rendering in a single low-cost plastic package possible for a wide range of applications. Moreover it leads to a substantial reduction of the main memory bandwidth requirement, which may drop from 400 MByte/s to 50 MByte/s which in turn makes it possible to integrate the architecture in a traditional PC or workstation.

19 Speed requirements

So far speed has not been an issue of concern, in actual practice it is however critically important as opacity and color should be calculated for all 512^3 = 134 Mega samples to obtain a single view of a dense dataset. This results in 3.36 Giga samples for 25 views each second and 94 Giga multiplications for the simplest algorithm. The natural cycle time of the low-power multiplier in the 1μm CMOS process is about 20ns. Within 8 years one may expect an intrinsic increase of the speed of a CMOS chip with a factor of 6. The total amount of multipliers needed for the rendering of a dense 256^3 dataset using a super sampling factor of 2 in each dimension will hence be 94 10^9 20 10^{-9} 1/6 = 313 at the moment at which a real-time rendering of a dense dataset on a notebook will be possible at acceptable energy levels. It can be shown that a single chip can perform this task.

20 Architectural constraints

The problem of the energy efficient rendering of a dataset does not allow much controller overhead, be it for a sparse or a dense problem. This is a point in common with most Reduced Instruction Set Computers. It is however not allowed either to use long datapaths do deliver operands to the multipliers, as this would violate our assumption about the use of a small fraction of the available power radius $\mathcal{R}_{pd}$ for wiring. Separate load and store instructions should be avoided as well, as these bring operands to a general purpose location, whereas it is important to have the interpolation circuits placed close to the (8 sections of the) v-cache. It is also important to have registers for a 3D vector increment: $\mathbf{x} := \mathbf{x} + \Delta\mathbf{x}$, or two 2D vector increments, closely connected to the v-cache address-bus. Separate modes in which either two, 2D interpolations can be processed or one 3D interpolation, using appropriate vector incrementers, makes the architecture multi functional.

It is possible to implement a Single Instruction Multiple Data architecture with multiple processors each operating on a single ray of a SIMD-bundle at one common depth location in the screen space. There are however some disadvantages to this approach: 1) It will be necessary to continue with the computation of *opacity* and the *gradient* for *all* rays on the SIMD-bundle, even when only *one* becomes opaque, 2) There may be samples on rays in the SIMD-bundle which lie *outside* the dataset, whereas others may still be *inside* the dataset, 3) It is necessary to continue with the rendering of *all* rays in the SIMD-bundle as long as there is still *one* left for which the sample location is still visible.

The most important problem with the SIMD approach is however the virtual impossibility to load the voxel-data in parallel. This is due to the fact that the arbitrary view direction implies a more or less random order in which the voxel data should be loaded. The SIMD approach seems for this reason to be viable for rather small bundles of say 2x2 using address transition detection hardware for the distinct addresses used on each of the rays.

The issue of conflict-free access to the voxel data from the v-caches is fully solved when single rays are processed using random access to rather small v-caches.

21 General remarks

It is rather crucial that the on-chip v-caches are small, as they are read with a high frequency. The energy overhead related to the use of small v-caches is related to the overlap of the cache sub volumes in the main memory. This makes it necessary to increase the size of the main-memory

accordingly. Moreover it affects the requirements for the data transfer rate. This effect is much less severe when an additional 16x16x16 cache is used between the main memory and the 8x8x8 v-caches. All the caches introduced can be loaded with 4x4x4 bursts from the main memory to optimally support the sparsity of the dataset.

22 Conclusions

It has been shown how the energy complexity of a challenging algorithm can be computed. Moreover a realistic lower bound on the energy consumption has been established, which shows that real-time volume rendering engines, based on a single chip housed in a plastic package are feasible for sparse problems using a 1μm CMOS process. It was shown implicitly that the skewed memory access which plays a key role in the CUBE 4 architecture is not needed for low power operation, nor for reasons of speed. Real time solutions for dense problems will become feasible within say 6 to 8 years from the introduction of the 1μm CMOS process. The 2 Watt power consumption constraint was applied to the whole rendering engine, i.e. including the required access to any external RAM and including all on-chip effects like energy consumption due to arithmetic, energy consumption due to wiring, which were shown to be negligible. The paper has not resulted in a detailed signal-flowgraph for a volume rendering engine, instead constraints were been specified, in terms of the maximum allowable fraction of $\mathcal{R}_{pd}$, available for placement of arithmetic blocks, the maximum size of the v-cache and feasible techniques for off-chip communication. An important aspect of the actual proof that the energy complexity is bounded by the figures given, comes from the observation that it is considered to be no major design problem to find a layout such that a fraction of the $\mathcal{R}_{pd}$ bound of the arithmetic elements is used for wiring. Address transition detection techniques were indicated to be effective to decrease the energy consumption of the v-cache, which was shown to be the second significant contribution to the energy complexity of the volume rendering algorithm.

23 References

[1] H. Pfister, A. Kaufman and F. Wessels, "Towards a Scalable Architecture for Real-Time Volume Rendering", Proceedings of the 10th Eurographics Workshop on Graphics Hardware, volume EG95HW, pp 123-130. EuroGraphics Workshop Proceedings Series, 1995.

[2] G. Knittel, "A Scalable Architecture for Volume Rendering", *Proceedings of the 9th Eurographics Workshop on Graphics Hardware*, volume EG94HW, pp 58-69. EuroGraphics Workshop Proceedings Series, 1994.

[3] S.E. Molnar, J. Eyles, and J. Poulton, "Pixelflow: High-Speed Rendering Using Image Composition", *Computer Graphics*, 26(2) pp. 231-240, July 1992.

[4] J. Lichterman, "Design of a Fast Voxel Processor for Parallel Volume Visualization", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp 83-92. EuroGraphics Workshop Proceedings Series, 1995.

[5] TMS320C80 Multimedia Video Processor (MVP), Texas Instruments, Digital Signal Processing Products series, 1994.

[6] A.P. Chandrakasan and R.W. Brodersen, *Low Power Digital CMOS design*, Kluwer Academic Publishers, Norwell MA, 02061 USA and Dordrecht the Netherlands.

[7] A. Bellaouar and M.I. Elmasr, *Low Power Digital VLSI Design*, Kluwer Academic Publishers, Norwell MA, 02061 USA and Dordrecht the Netherlands.

[8] M. Bosma, J. Smit, and J. Terwisscha van Scheltinga, "Superresolution Volume Rendering Hardware", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp 117-122. EuroGraphics Workshop Proceedings Series, 1995.

[9] J. Terwisscha van Scheltinga, J. Smit, and M. Bosma, "Design of an on-chip reflectance map", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp. 51-55. EuroGraphics Workshop Proceedings Series, 1995.

[10] D.J. Kinniment, J.D. Garside, and B. Gao, "A Comparison of Power Consumption in Some CMOS Adder Circuits", *Proceedings of PATMOS*, Eds C. Piguet, W. Nebel, pp. 106-118. ISBN 3-8142-0526-X

[11] J. Smit and J.H. Huisken, "On the energy complexity of the FFT", *Proceedings of PATMOS*, Eds C. Piguet, W. Nebel, pp. 119-132. ISBN 3-8142-0526-X

[12] M. Bentum, *Interactive Visualization of Volume Data*, Phd. Thesis, University of Twente. ISBN 90-9008788-5.

[13] M. Bosma and J. Terwisscha van Scheltinga, *Efficient Super Resolution Volume Rendering*, Masters thesis, University of Twente, August 1995.