The Role of Power Dissipation and Locality of Reference in the Specification of High Performance Graphics Algorithms

J. Smit, M.J. Bentum, M.M. Samsom
University of Twente
Laboratory for Network Theory and VLSI Design
P.O. Box 217
7500 AE Enschede
The Netherlands
jaap@nt.el.utwente.nl

Abstract

The amount of power dissipated by the implementation of an algorithm, for instance in the form of a dedicated chip-set, is considered to be one of the most important constraints for the selection of a high performance graphics algorithm. This is due to the fact that the realization of computational capability within the reach of one Tera operations per second is non-practical with general purpose CPU-chips. The case study of a high performance surface visualization engine is used to introduce the reader with the aspect of power dissipation in relation to computational power. We introduce a low-power 'parallel datapath' RISC processor, based on a highly efficient mapping of locality of reference in the algorithm onto silicon. A subsequent classification is made for various high performance graphics algorithms.

Introduction

In this paper we will address the role of power dissipation and locality of reference when developing a high performance graphics engine. As example we use a surface rendering engine.

A Surface Rendering Example

A surface rendering engine will reproduce the projection of a 3D scene on a 2D display unit. The common technique to solve this problem is to fill the projection of patches on the display screen with a constant intensity value. A much more realistic image can be obtained with relatively large patches when the intensity values are not taken to be constant. For instance the graphics processor designed by our group [1] in collaboration with the CWI [2], can solve a constant, first order and second order forward difference equation, between two arbitrary points A and B with an index from 0 ... 1023, given the value of the Intensity (I) and its first (DI) and second derivative (DDI), using a systolic engine built from 1024 identical processing elements. In Figure 1 the algorithm is shown.

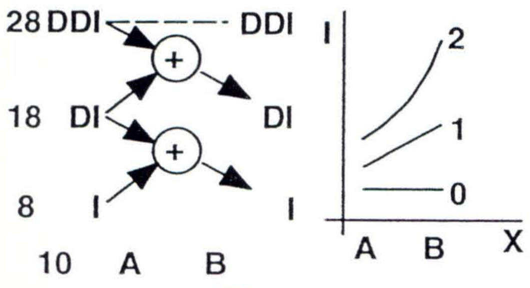

Fig. 1

The new values for I, DI and DDI are, just like the decremented values of A and/or B calculated in the first generation design of the high performance surface rendering algorithm, using a multiplexed 3 x 12 bit ALU with pipelined ripple carry adders. A dedicated controller moves the values of I, DI, DDI, A and B to and from a bank of registers located in the current processing element as well as as in the next one.

A floorplan of the first generation design is given in Figure 2. The areas with a dark shading represent the ALU, the medium shaded areas are occupied by registers. The light shaded area is the controller. The black line represents one wire in the interprocessor bus.

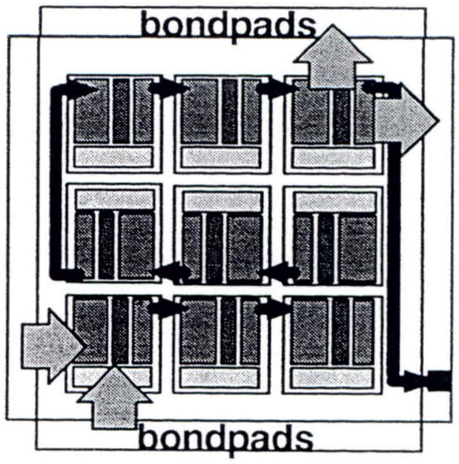

Fig. 2

The chip was designed for operation at 80MHz, it should be noted however that this speed had to be derated, due to the 5.2 Watt of power dissipated at the target speed, as most available packages could dissipate no more than a mere 2 Watt for reasonable junction temperatures. Given these results, we realized that power dissipation was the main obstacle to extend the capabilities of the design.

Power dissipation

The amount of power dissipated in a VLSI-chip is: .5 x Vcc^2 x Cpd x f, where Vcc is typically 5 Volt, Cpd is the average sum of capacities being charged or discharged and f the frequency of operation. It is useful to split the value of Cpd into a part due to arithmetic, one due to register usage, and one due to interconnect. The 3 x 12 bit multiplexed ALU in the first generation design was not fast enough for the required 80 Mhz operation, or 12.5 ns cycle time, as the time for one bit-addition is about 1.5 ns. The inclusion of pipeline registers in the ALU and the additional control logic for the registers, made a 12 ns design feasible. Note that multiplexing is a slow-down operation, whereas pipelining is a speed-up operation.

Multiplexing caused operands to be moved 5 times over the chip from the registers to the ALU of the current and the next processor. The use of pipeline registers placed in the control lines of the registers made the registers relatively large. As a consequence they were also relatively 'far' away from the ALU, resulting in a relatively high wiring capacity.

The second generation design

We have used another technique for the second generation design. Using a parallel datapath approach, we took advantage of the fact that two n-bit additions can be executed in slightly more than n times the carry propagation speed, as the sum-bits in a ripple carry adder are already after one sum-propagation time available, as shown in Figure 3.

So any arithmetic expression with n-bit wide additions (+/– operators) nested k levels deep can be calculated in n x T_{carry} + k x T_{sum}.

We used this observation for the design given in Figure 4. Five identical PE-sections were cascaded without any intermediate registers. These units compute five output values in 60 ns, resulting in a computation rate of one value every 12 ns. The power dissipated by this design can be expressed in full-adder equivalents, using the following equivalence rules expressed in full adder equivalents:

1) A fast (12ns) register => .5 FA Equivalents
2) A slow (60ns) register => .2 FA Equivalents
3) One wire spanning the width of a FA => .02 FA Equivalents
4) One wire spanning the height of a FA => .05 FA Equivalents

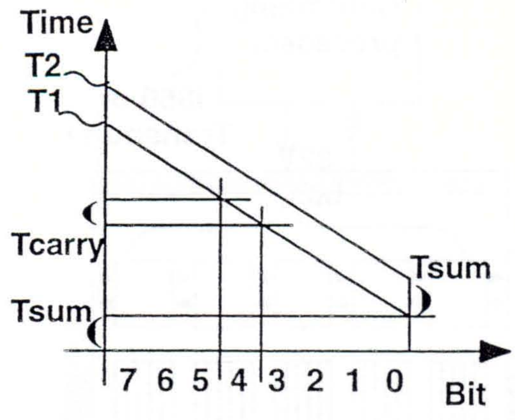

Fig. 3

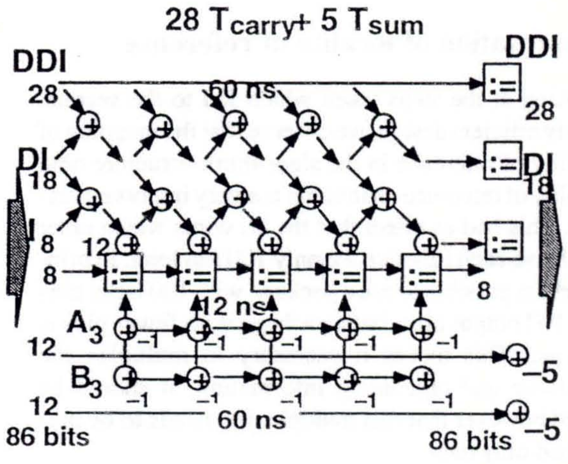

Fig. 4

Using these figures a good mapping should be found from the algorithmic structure in Figure 4 to a layout on silicon. A good mapping is already given in Figure 4, provided that the most significant bits of the full adders for DI and I are aligned, i.e. one should not place these 28 and 18 bit adders under each other but immediately adjacent to each other.

The overall graphics processor is realized using the floorplan of Figure 5. Individual PE-slices are 2 full adders wide. Five slices are grouped into units of 500 µm wide and 6.5 mm high in a 1.5 µm process. A total of 100 PE-slices can be realized on a single chip, dissipating less than 2Watt. This is a considerable gain compared to the first design, where 9 PEs dissipated 5.2 Watt.

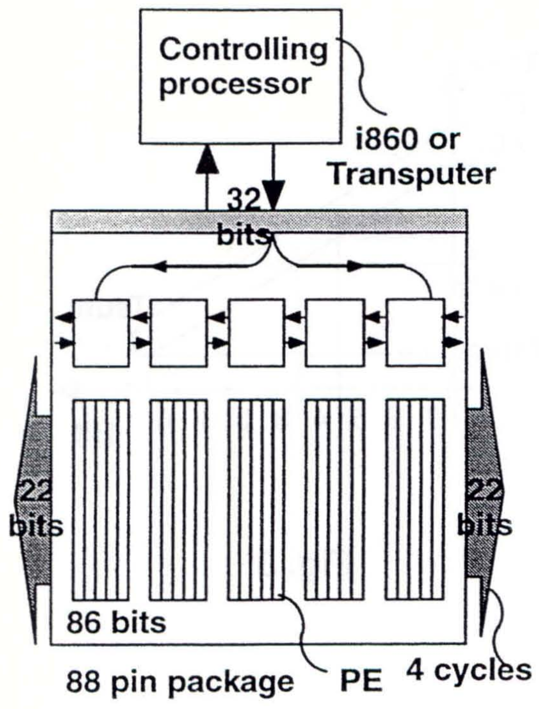

Fig. 5

Preservation of locality of reference

Looking at the steps taken which led to the second, highly efficient design, we observe that the mapping of locality of reference in the algorithmic structure onto locality of reference on the chip is a very important feature. This had as effect that the 86 wires which enter and leave each PE-slice are only 100 μm long. In principle it is possible to use a package with 100 input pins and 100 output pins, but a package with fewer pins is cheaper. This makes it necessary to multiplex the incoming and outcoming information. It should be noted however that this multiplexing needs to be performed only once.

In contrast we see the use of a 3mm long interprocessor distance, which carries in essence the same information, using a high (5x) multiplex rate. A calculation of the resulting power dissipation can be found in [3].

Maximum arithmetic capabilities

The arithmetic capabilities, expressed for instance in terms of the maximum number of 8-bit additions, executed on a 7 x 7 mm^2, 1.5 μm CMOS chip, will now be calculated, assuming that 5 slow registers are used in conjunction with 2 8-bits arithmetic units, to store intermediate results of calculation. Six such units fit in an area of 1 mm^2. So at most 2 x 6 x 7 x 7 = 588, 8-bit full adders with accompanying registers would fit on a 7 x 7 mm^2 chip. These arithmetic units could be clocked at a rate of one operation in 20 ns even if the arithmetic operations are nested up to five levels, so the overall computational power of a 7 x 7 mm^2 1.6 μm chip is about 50 x 10^3 x 600 = 30 Giga operations per second. The value of Cpd for the full adder is 500 fF, this gives a total dissipated power of: 1/2 x 5^2 x 600 x 24 x 500 10^{-15} x 50 10^6 = 4.5 Watt. Half of this amount will under normal circumstances be dissipated if we assume a probability of 50% that any bit will change.

Efficiency of algorithm mappings

The multiplexed ALU in the first generation design was constructed from 3 12-bit wide adders. The instruction set used five cycles of this ALU to solve the forward difference equation for a single step along the scan-line. This gives an arithmetic power-complexity of 900 full-adder equivalents for the arithmetic of 5 PEs. The registers, excluding the pipeline registers have an power-complexity of 660 full-adder equivalent for 5 PEs. The mapping of the arithmetic on registers is much more efficient in the second generation design, as all word-lengths used are minimal. Moreover the detection of the interval A–B can be done with two 3 bit comparators only, as the global location of the interval may be computed once per section of 5 PEs, using two 12 bits comparators. The registers used to store intermediate results in the first generation design are fully absent in the second generation design. This has as net effect that the amount of power dissipated in the second generation design, due to register usage is 144 Full Adder power-equivalents, whereas this same figure is 660 Full Adder power-equivalents in the first generation design. This shows that one may obtain a more efficient implementation of a given algorithm using adequate word-lengths and locality of reference aspects.

It should be noted however that locality of reference in the layout of the algorithm plays an even more important role, as can be seen by the difference between both designs concerning the effect of the wiring capacitance on the total amount of power dissipated by the algorithm.

Aspects of graphics architectures ranging from general purpose to dedicated

We will now discuss some of the performance criteria for various forms of chip-realization in terms of the desired performance of a given algorithm. The arithmetic capability of the chip will be taken as a reference point with respect to which we may argue that further improvement is not possible.

The effectivity of a general purpose solution depends much on properties of the target algorithm, like:

1 Is it desired to realize the worst-case performance or should the graphics algorithm realize the desired performance for some average case?

2 Is the IO-bandwidth of the chip sufficient, i.e. does the algorithm require random data from an external RAM?

3 Should special techniques be used, like dedicated RAS/CAS selection for fast RAM access?

The parallel-datapath RISC-engine

One type of general purpose processor might be a parallel datapath RISC processor. Such a processor would fetch a word from the external memory and decode and execute all instructions fetched in parallel, using 8 decoders without internal registers. When a 128-bit wide memory interface is used in conjunction with a 16-bit instruction format, it could execute at most 8 128-bit instructions at a time.

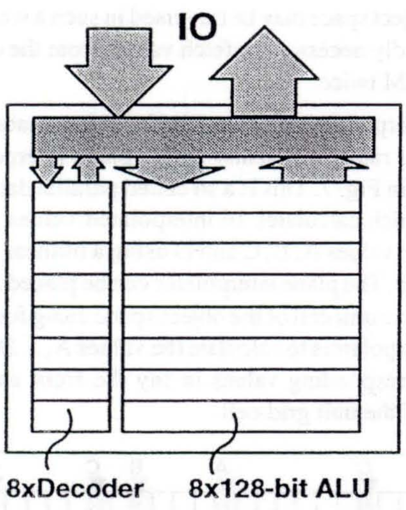

Fig. 6

These instructions might be partitioned into 8-, 16-, 32-, 64- or 128-bit units and be configured to execute a fast carry select addition or subtraction, an nxn multiplication, a 2nxn division etc. So one instruction fetch might be used to execute 8 x 16 = 128 8-bit add-subtract instructions or 8 x 4 = 32 32-bit add-subtract instructions in say 20 ns. The architecture introduced can reach a considerable speed of about 1.6 Giga 32-bits add-subtract instructions per second. This is especially favorable compared to modern RISC processors which use considerably more power for a fraction of the workload, in a much more advanced process. The final bottleneck for a parallel datapath RISC engine will be the IO-bandwidth, and its restricted ability to output results using the IO subsystem. This bottleneck is however inherent to the idea of a general purpose machine.

The loop-unrolling engine

The ability of the parallel datapath engine to control not more than 8 arithmetic and logic units each executing one sequential instruction from the instruction stream, may be seen as a disadvantage. An instruction set with special features for loop-unrolling might be a solution to this problem. The prefetch mechanism would repeatedly fetch instructions from a LOOP ... ENDLOOP construct, until either all ALUs available have got instructions and the repeated LOOP execution can start, or the LOOP is sufficiently rolled out. Using this way of prefetching one can set-up a quite efficient repeated execution of the instructions to be executed within the LOOP... ENDLOOP construct.

Locality of reference within programs

Even the loop-unrolling engine will not be able to reach its performance limit when the graphics program to be executed needs abundant IO-bandwidth, as ultra-high bandwidth requires either an unrealistic pin-count or ultra-high speed IO-pads. On-chip memories may be used to increase the IO-bandwidth of an algorithm, either as implicit cache or as explicit intermediate storage, to reduce the off-chip IO-rate. One extreme is to put all RAM on-chip. This may be a problem as the advanced technology makes off-chip RAM always faster and larger than on-chip RAM. An alternative might be to put a moderate size graphics processor on the mask-set of an external RAM, to fully exploit the high (e.g. 512-bit) on-chip bandwidth. The proposed merge of on-chip RAM and a graphics processor is however frequently impossible, as the chip-foundry has specialized production lines for RAM and for logic, such as CPUs.

These issues imply that we should seek for locality of reference in the application program and seek for opportunities to include the mapping of the algorithm onto silicon as part of the compilation process, when extreme speed is of importance.

Statistics of graphics engines

A true Von Neumann computer adapts itself, by its very nature always to the statistics of the given algorithm. The aspect of random access in the instruction stream and the datastream is the most helpful property in this respect. It is this same aspect however which limits the performance of architectures with a Von Neumann structure, as the IO-bandwidth of the chips used, be it the memory chips or the CPU is frequently the limiting performance factor for Von Neumann machines. The classical technique used to improve the performance of such a system: acceleration of the ALU, has as main effect that the power dissipation goes up considerably more (sometimes as much as two orders of magnitude) than one would expect on the basis of the increased processing speed. The architectures discussed so far can be used to run algorithms at a much higher speed than classical architectures. This high speed can even be obtained at power dissipation levels which are quite attractive. Most algorithms loose on such architectures their ability to adapt to the statistics of the algorithm, instead they are capable to execute a graphics algorithm at worst case performance conditions.

Specific algorithms:

The surface rendering example

One of the techniques used to obtain the desired performance level is the use of instructions which are fed over an array of processing elements as shown in Figure 4. An implementation of a systolic engine in which the instructions may either flow from left to right or the other way round is shown in Figure 5. This architecture may also be used to let instructions jump over PE-blocks, when it is known that an instruction would not be executed at any of the PEs within the block. This aspect makes it easier to let the proposed engine borrow processing power from adjacent PE blocks. Borrowing processing power from adjacent scan-lines makes it necessary to include at least multiple pixel intensity accumulation registers as well as a provision to indicate in the instructions which scan-line should be affected.

The algorithm used in the architecture of Figure 4 for the shading of 2D patches from 3D data, requires that the values of I, DI and DDI are computed at a worst-case rate of 200 values per scan-line. This makes this algorithm very IO-dependent. Moreover the actual calculation of these values is a big problem as it requires high precision (floating point) arithmetic. An algorithm which would locally subdivide a 3D patch into smaller ones until the patch could be shaded with first order (Gouraud) shading techniques can use the aspect of on-chip locality of reference much better. Multiple, interpolated look-up tables may be used in such a variant of the algorithm to calculate intensity values for the patches involved.

The mapping of the algorithm on the VLSI-chip becomes very inhomogeneous when the description of the 3D patches would be done with floating-point arithmetic, whereas all intensity values would be calculated with 8-bit accuracy. This is not a problem for a dedicated chip, but it is likely to be a problem for the general purpose solutions shown before. An alternative hardware realization might be here a "Sea of arithmetic building blocks", which could be programmed using a static program, downloaded from an external EPROM, like a field programmable gate array. A library with primitives ranging from floating point units to simple 8-bits arithmetic elements may be used to configure the chip for the algorithm.

The volume rendering algorithm

Medical imaging applications of the volume rendering algorithm usually start from a 3D set of say 256^3 or 512^2x64 opacity and color values located on a grid in the object-space. The color values are precalculated with the common light equation. Opacity values are assigned on the basis of a tissue classification step. It is in this way that a user may select skin and bone to be transparent, but brain opaque etc.

The given color and opacity values are sampled at the grid-points of the display space and interpolated for subsequent use in a non-linear difference equation, which solves the propagation of light in a non-homogeneous medium.

Locality of reference in the object space can be exploited in the following ways:

1 The way in which an external RAM is accessed can be optimized in such a way that the values which may be reached from a given row, lie in a subspace of the object space with almost equal sizes in X, Y, Z.

2 The object space may be traversed in such a way that it is hardly necessary to fetch values from the external RAM twice.

3 The interpolation necessary in the object space may be done rather efficiently with a plane interpolator shown in Fig. 7. This is a so called parallel datapath unit which calculates 16 interpolated values given the four values A, B, C and D using a bilinear interpolation. The plane interpolator can be placed at any depth in a unit cell of the object space using four linear interpolators to calculate the values A, ... D from the corresponding values in say the front and the back of the unit grid-cell.

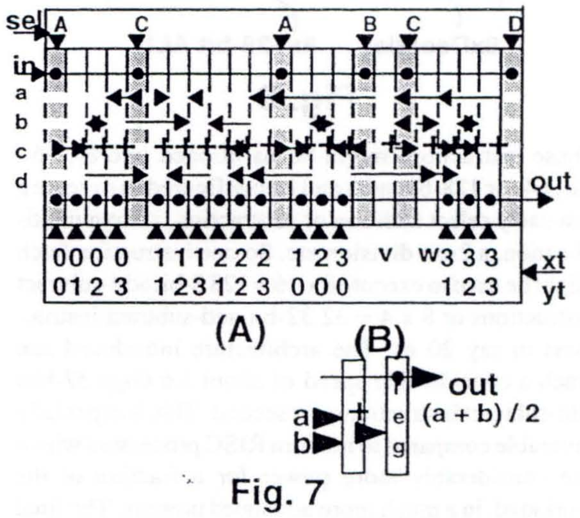

Fig. 7

Locality of reference in the display space can be exploited in the following ways:

1 The calculation of opacity and color values along a ray, the composition operation, can be done either front to back or back to front. These algorithms use local accumulators. The balanced tree version of the composition operation is less regular and hence does not map locality of reference in the algorithm to locality of reference on the chip. The differences are however slight in this case as the composition opera-

tion needs large multipliers which have a lot of internal locality of reference but the external components are relatively far away anyhow.

2 The fact that the cross-section of a ray in the display-space with the object space gives a ray in the object space, leads to a loss of locality of reference in the object space. The use of an n x n bundle of rays overcomes this problem. It should be noted however that this requires a total of n^2 accumulator values, which have to be stored in an intermediate memory.

3 To minimize the on-chip traffic even more we process all interpolation planes which span a single elementary cell in the object space. This lowers the rate at which accumulated values of opacity and color should be accessed.

4 The generation of the many object space addresses for all locations in the display space can be done using a differential analyzer. This requires however calculations with a high (21 bit) precision. Instead one may use the locality of reference in the display space to generate all addresses in the bundle-plane all at once using a plane interpolator, given at least four points calculated with a differential analyzer. The interpolation can be done with a highly reduced precision, as the final resolution which we want to maintain corresponds with an interpolation factor of 4 on the object-space grid.

Statistics of graphics algorithms

Three-dimensional visualization algorithms like those used in flight simulators, are frequently designed to handle an object space which is considerably larger than the object space typically used in volume rendering. This makes high accuracy, say 32-bit, (floating point) arithmetic necessary. The large object space is however necessarily sparse. This has as effect that these algorithms rely much more on random access. The capabilities of those algorithms are however also much more restricted, as texture mapping is frequently not supported. Hidden surface removal runs frequently with little or no hardware support. It will be clear from the material discussed in this paper that those algorithms run normally with a relatively poor performance, which is highly restricted by the degree of sparseness of the object space. A sparse implementation of the volume rendering algorithm, which fills the 3D object space with a set of small volumes which contain all visible parts of the scene might be used to obtain a realistic visualization of large, complex scenes with improved performance and better realism through additional object detail. The hardware support for hidden surface removal, transparency and texture for fine details is one of the features which the volume rendering algorithm adds to the general technique of handling a large sparse object-space.

Discussion

We hope that the contents of this article will contribute to a further understanding of the importance of locality of reference in algorithms and the ways in which this can be mapped onto silicon to obtain a performance gain of some orders of magnitude. We are convinced of the fact that any method to increase the performance of (graphics) algorithms will be bound by aspects of the reduction of power dissipation, as power dissipation is a very fundamental effect, which becomes more and more important when we want to extent the computational limits beyond present limits. The ways in which the desired performance limits can be reached are in no way the only ones. This is why we have given a description of the main lines of thought used by us to develop high performance graphics engines. In addition we have presented alternative ways to exploit similar performance levels using new programming techniques.

References

[1] J.A.K.S. Jayasinghe et al., "Two-level Pipelined Systolic Array Graphics Engine," IEEE Journal of Solid-State Circuits, Vol. 26, no. 3, March 1991, pages 229-236

[2] P.J.W. ten Hagen, A.A.M. Kuijk and C.G. Trienekens, "Display Architecture for a VLSI-based Graphics Workstation." internal report no. CS-R8637, CWI, Amsterdam, 1986

[3] J. Smit, M.M. Samsom and H. Snijders, "High Speed Surface Rendering of 3D Images Using a Novel Chip-Design Methodology," in proceedings of the ProRISC IEEE Benelux Workshop on Circuits, Systems and Signal Processing, Houthalen Belgium, March 24-25 1993, pages 227-232