A Fast, Cache-Aware Algorithm for the Calculation of Radiological Paths Exploiting Subword Parallelism

Mark Christiaens, Bjorn De Sutter, Koen De Bosschere,
Jan Van Campenhout and Ignace Lemahieu

Electronics and Information Systems Department
University of Gent, Belgium
{*mchristi,brdsutte,kdb,jvc,il*}*@elis.rug.ac.be*

Abstract

The calculation of radiological paths is the most important part in statistical positron emission tomography image reconstruction algorithms. We present a new, faster algorithm which replaces Siddon's. Further code transformations on this algorithm prove to be beneficial in a Maximum Likelihood - Expectation Maximization reconstruction algorithm and the result is perfectly suitable for an implementation that exploits the VISual instruction set from Sun or other modern architectural extensions providing subword parallelism. The final speed-up achieved with this new algorithm and its subword parallel implementation is 13. Though smaller data formats are used in subword parallelism, the resulting images are as good as the original ones.

1 Introduction

The reconstruction of an image from the data obtained from a tomographic scan using statistical algorithms requires a measure of the contribution of the various parts of the body to the radiation received by a sensor pair of the PET-scanner (Positron Emission Tomography). This measure is obtained by calculating the radiological path: the line-integral of the local intensity of radiation in the body along the ray connecting the sensors. During a full image reconstruction, radiological paths are calculated for millions of rays corresponding to different sensors of the scanner, positioned at different angles. This calculation is the most time-consuming part of image reconstruction algorithms and Maximum Likelihood - Expectation Maximization (ML-EM) [6] in particular. ML-EM is not the fastest reconstruction technique available today.

Preprint submitted to Elsevier Preprint

Nevertheless, it is a representative of a whole class of statistical PET-image reconstruction algorithms. The proposed techniques are valid for each of these.

Known algorithms for this calculation, such as the one Siddon [7] proposed, can be performed in parallel on multiprocessor machines. Each processor then calculates a different set of rays. It has been shown that this parallelization is very efficient [10].

Modern architectures, e.g. SPARCv9 [2] and PA-RISC 2.0 [1], are not only suited for multiprocessor systems, but nowadays they offer a new form of parallelism, namely subword parallelism. In this form, smaller data formats, such as bytes, are used as subwords that fill one word and on which one instruction is able to perform the same operations in parallel. While this subword parallelism was introduced mainly to increase performance in multimedia applications, it has been used in different domains, such as CRCs (Cyclic Redundancy Check) and RAID-systems (Redundant Array of Inexpensive Disks) [9].

In this contribution we will show that statistical PET image reconstruction can significantly benefit from subword parallelism. A new, incremental algorithm is proposed, that replaces Siddon's algorithm. The fastest version of our algorithm in the context of PET image reconstruction, where radiological paths of parallel rays are calculated, is created using loop and if-then-else transformations. These transformations are at the same time the basis for exploiting SIMD (Single Instruction Multiple Data) or subword parallelism.

In the following sections, we will propose this new algorithm and the successive transformations step by step.

2 Calculation of the Radiological Path

The reconstruction of an image from the data obtained from a tomographic scan requires a measure of the contribution of the various parts of the body to the radiation received by a sensor pair. This is done by calculating the radiological path.

2.1 Definitions

The radiological path is the line-integral of the local intensity of radiation in the body along the ray connecting the sensors. If we divide the space, in which the image is to be reconstructed, into pixels, this calculation is equivalent to finding those pixels crossed by the ray and the distance traveled in every such pixel. Since this data only depends on the geometry of the scanner, one might

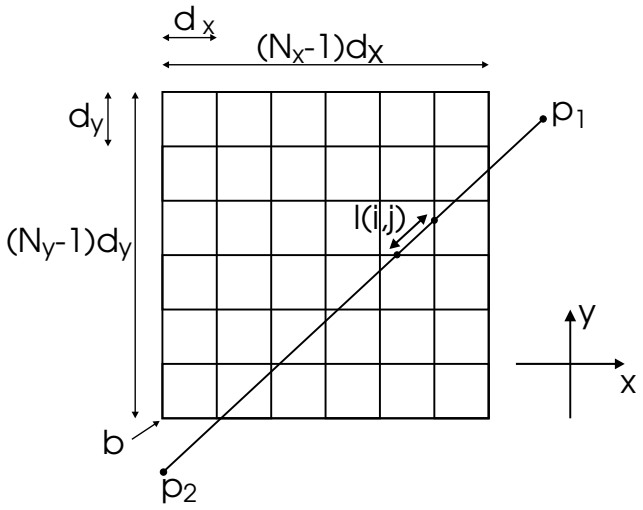

Fig. 1. Reconstruction domain

try to store it for later reuse. This is infeasible for all but the smallest of images, due to the sheer magnitude of the data. Therefore it is necessary to regenerate this data on the fly.

We describe our algorithm for 2D images for the sake of clarity; it is easily converted to 3D. In what follows we will approximate the image by a rectangle, its sides parallel to the x and y axes and divided into smaller rectangular pixels, as shown in Figure 1. Let $\mathbf{b} = (b_x, b_y)$ be the lower left corner of the rectangle, N_x and N_y be the number of vertical and horizontal lines resp. in the grid and d_x and d_y be the dimensions, in the x and y direction resp., of the pixels. We shall use indices i, j to refer to individual pixels. The i index runs along the x axis, and the lower left corner pixel has index $(0, 0)$. A ray will be represented by two points: $\mathbf{p1} = (p1_x, p1_y)$ and $\mathbf{p2} = (p2_x, p2_y)$. We define the density of a pixel (i, j) as $\rho(i, j)$. The distance covered by a ray in the pixel (i, j) is $l(i, j)$, where $l(i, j) = 0$ for pixels not being crossed. The radiological path is then defined as

$$d_{p_1 p_2} = \sum_{i,j} \rho(i, j) l(i, j). \tag{1}$$

Other approaches to approximate the contribution of pixels to the radiation measured are used, but are less accurate or much slower. Using radiological paths is a good compromise between speed and accuracy.

2.2 Siddon's Algorithm

Equation (1) could be evaluated by simply summing for *all* (i, j). This would be unwise, as pointed out by Siddon, since most $l(i, j)$ are zero. It is more effi-

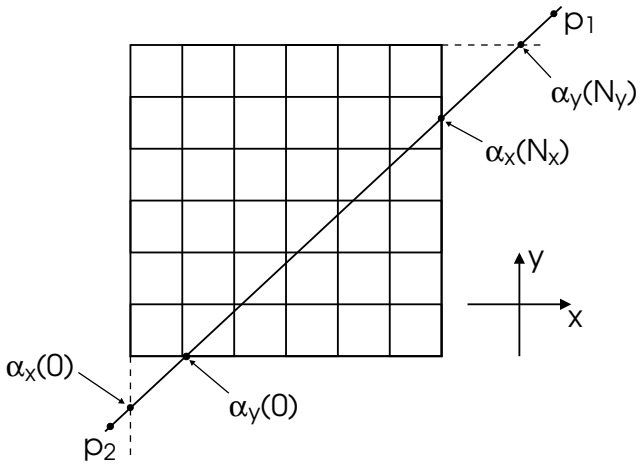

Fig. 2. Calculation of the entry and exit point

cient to evaluate $d_{p_1p_2}$ by following the path of the ray. We therefore construct a parameter representation of the ray

$$\mathbf{p}(\alpha) = \mathbf{p_1} + \alpha(\mathbf{p_2} - \mathbf{p_1}) \tag{2}$$

This allows us to determine the distance travelled through a pixel by multiplying the α-"distance" with a conversion factor.

In what follows, we shall assume that this is a generic ray, i.e., that it is neither parallel to the x axis, nor to the y axis. Those rays that are parallel are easily implemented and form only a small fraction of the execution time of a complete image reconstruction. Following Siddon, we calculate the entry point ($\alpha = \alpha_{min}$) and the exit point ($\alpha = \alpha_{max}$) of the ray w.r.t. the reconstruction rectangle (Figure 2). We need the parameter values of the intersections of the ray with the 2 outer sides perpendicular to the x axis ($\alpha_x(0)$ and $\alpha_x(N_x)$) and the y axis ($\alpha_y(0)$ and $\alpha_y(N_y)$). We have

$$\alpha_x(0) = \frac{b_x - p1_x}{p2_x - p1_x}, \tag{3}$$

$$\alpha_x(N_x) = \frac{b_x + (N_x - 1)d_x - p1_x}{p2_x - p1_x} \tag{4}$$

and a similar formula for $\alpha_y(0)$ and $\alpha_y(N_y)$.

Then the entry and exit parameters are given by

$$\alpha_{min} = \max\{\min\{\alpha_x(0), \alpha_x(N_x)\}, \min\{\alpha_y(0), \alpha_y(N_y)\}\}, \tag{5}$$

$$\alpha_{max} = \min\{\max\{\alpha_x(0), \alpha_x(N_x)\}, \max\{\alpha_y(0), \alpha_y(N_y)\}\}. \tag{6}$$

The number, n, of pixel-boundaries crossed, outer limits included, can be calculated as the sum of the intersections with boundaries perpendicular to the x and y axes:

$$n = i_{max} - i_{min} + 1 + j_{max} - j_{min} + 1, \tag{7}$$

Here i_{min} and j_{min} are the indices of the first pixel boundary the ray will cross *after* entering the rectangle and i_{max} and j_{max} are the last pixel boundary that is crossed *including* the border of the rectangle. If $p1_x < p2_x$, we find:

$$i_{min} = \left\lfloor N_x - \frac{b_x + (N_x - 1)d_x - \alpha_{min}(p2_x - p1_x) - p1_x}{d_x} \right\rfloor, \tag{8}$$

$$i_{max} = \left\lfloor \frac{p1_x + \alpha_{max}(p2_x - p1_x) - b_x}{d_x} \right\rfloor. \tag{9}$$

The equations for $p1_x > p2_x$ and for j_{min} and j_{max} are similar.

Siddon continues by creating arrays a_x and a_y of αs corresponding to intersections with pixel sides perpendicular to the x and y axes, resp. . These are merged into one larger array a of ascending values. This array is then used to calculate the pixels that are being intersected as follows. Two consecutive values α_{m-1} and α_m are taken and the mean value is calculated:

$$\alpha_{mid} = \frac{\alpha_m + \alpha_{m-1}}{2}. \tag{10}$$

Using this value the pixel's coordinates (i, j) are given by:

$$i = \left\lfloor 1 + \frac{p1_x + \alpha_{mid}(p2_x - p1_x) - b_x}{d_x} \right\rfloor \tag{11}$$

and a similar equation for j. Now we can compute the distance the ray travels through this pixel

$$l(i, j) = d_{conv}(\alpha_m - \alpha_{m-1}) \tag{12}$$

with

$$d_{conv} = \sqrt{(p2_x - p1_x)^2 + (p2_y - p1_y)^2}. \tag{13}$$

From this the radiological path can be computed.

This approach has two major disadvantages:

- The calculation of the indices of the pixels through which a ray passes is very complex and time-consuming.
- Four major inner loops are necessary.

So far, we have formulated Siddon's approach. Now we will propose ours which is considerably faster and makes an SIMD implementation possible.

2.3 The Incremental Algorithm

Siddon collects information on the whole traversal in arrays and uses these afterwards to calculate the radiological path. We walk through the image pixel by pixel and calculate the radiological path immediately as follows.

First we need to calculate the first pixel the ray intersects with. In what follows, we will suppose that we are dealing with a ray that enters through a boundary parallel to the x axis i.e. $\alpha_{min} = \alpha_x(0)$ or $\alpha_{min} = \alpha_x(N_x)$. The i pixel index is then given by

$$i_{start} = \begin{cases} 0, & \text{if } p1_x < p2_x \\ N_x - 2, & \text{if } p1_x > p2_x \end{cases} \tag{14}$$

and the j index by

$$j_{start} = \begin{cases} \left\lfloor \frac{p1_y + \alpha_{min}(p2_y - p1_y) - b_y}{d_y} \right\rfloor, & \text{if } p1_y < p2_y \\ \left\lceil \frac{p1_y + \alpha_{min}(p2_y - p1_y) - b_y}{d_y} - 1 \right\rceil, & \text{if } p1_y > p2_y. \end{cases} \tag{15}$$

At this point (Figure 3) we have just entered the rectangle, hence $\alpha = \alpha_{min}$. There are two possible next steps, i.e. we cross a boundary perpendicular to the x axis or one to the y axis. Hence we compute α_x and α_y, the α parameter values of these crossovers, as follows

$$\alpha_x = \alpha_{min} + d_{\alpha_x}, \tag{16}$$

$$\alpha_y = \frac{\left\lceil (y1 + \alpha_{min}(p2_y - p1_y) - b_y) d_y^{-1} - 1 \right\rceil d_y + b_y - p1_y}{p2_y - p1_y} \tag{17}$$

with

$$d_{\alpha_x} = \left| \frac{d_x}{p1_x - p2_x} \right| \tag{18}$$

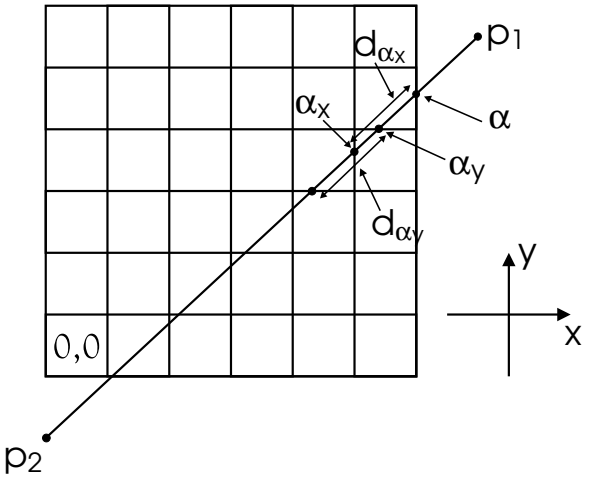

Fig. 3. Incremental values

if $p1_y > p2_y$. Similar equations for $p1_y < p2_y$ and d_{α_y}. The values of d_{α_x} and d_{α_y} are the distances measured in α-units from one intersection with a pixel boundary perpendicular to the x axis and the y axis resp. to another.

We will now assume that the image is stored as an array, b, and that the pixels (i, j) and $(i + 1, j)$ are stored in adjacent positions in memory. The index w of a pixel (i, j) in the array is then:

$$w = j(N_x - 1) + i. \tag{19}$$

In order to perform only one integer addition to calculate the next pixel we define the following variables:

$$w_x = \begin{cases} -1 \text{ if } p1_x > p2_x \\ 1 \quad \text{if } p2_x > p1_x \end{cases}, \tag{20}$$

$$w_y = \begin{cases} N_x - 1 \text{ if } p1_y > p2_y \\ 1 - N_x \text{ if } p2_y > p1_y \end{cases}. \tag{21}$$

Therefore w_x and w_y are the steps one has to take, in b, when an x boundary or a y boundary is crossed. The index in the array of the first pixel crossed is given by

$$w_1 = j_{start}(N_x - 1) + i_{start}. \tag{22}$$

All that is left is to specify an iteration scheme that is repeated once for each new pixel and that calculates the radiological path (starting value $d_1 = 0$) along the way. This is quite simple since we only need to check whether $\alpha_x < \alpha_y$

to know which boundary is crossed next. From this information it is easy to adjust all the variables to allow another iteration:

$$\begin{cases} d_i &= d_{i-1} + (\alpha_y - \alpha_{i-1})b[w_{i-1}], \\ \alpha_i &= \alpha_{y_{i-1}}, \\ \alpha_{y_i} &= \alpha_{y_{i-1}} + d_{\alpha_y}, \\ w_i &= w_{i-1} + w_y \end{cases} \tag{23}$$

if $\alpha_{y_{i-1}} < \alpha_{x_{i-1}}$ and similar adjustments if $\alpha_{y_{i-1}} \geq \alpha_{x_{i-1}}$. This process is repeated n (eq. 7) times after which d_{n+1} is multiplied with d_{conv} (eq. 13) resulting in the radiological path $d_{p_1p_2}$. A new ray can be processed.

Our algorithm is faster mainly because of the simpler index calculations (eq. 23), since integer addition is the only operation used for this part of the algorithm. On UltraSPARC-II this is advantageous for at least three reasons, which are relevant for most CPUs. In order of importance:

(1) The Siddon algorithm uses floating-point to integer conversion. This is necessary because integer registers are used for indirect memory accesses, i.e. $b[w]$ in the inner most loop. Because the UltraSPARC-II architecture [4,11] does not provide direct moves from floating-point to integer registers, this is an extremely time-consuming operation. All moves from floating-point to integer registers are implemented by store/load pairs. Forwarding between store buffers and loads is yet to be implemented so these moves must pass through main memory. Our incremental algorithm uses this type of conversion once per ray, instead of once per pixel.
(2) The Siddon algorithm uses floating-point arithmetic for index calculations. All other calculations are floating-point as well. This means they have to share the limited floating-point resources (ALUs). The index calculations in the inner loop of the incremental algorithm are integer arithmetic. Since integer resources are available, these calculations do not take extra time: they are performed in parallel with the floating-point arithmetic.
(3) Fewer floating-point registers are used, leaving more room for code optimization by the compiler.
(4) The smaller inner loop facilitates loop unfolding.

3 Introducing Subword Parallelism

The algorithm, in its present form, is still not suitable for exploiting subword parallelism. This can easily be seen in eq. 23, the inner loop of our algorithm.

```
for each angle in reconstruction do
 for each ray at angle do
  initialize calculation of the radiological path
  for each pixel on path do
   if (α_{y_{i-1}} < α_{x_{i-1}})
    advance up or down (eq. 23)
   else
    advance sideways (analogue to eq. 23)
   fi
  od
 od
od
```

Fig. 4. Code structure of one step during iterative image reconstruction in ML-EM

```
for each angle in reconstruction do
 for each 4 parallel rays at angle do
  initialize RP-calculation 4 rays
  for each pixel on paths do
   if − then − else_1
   if − then − else_2
   if − then − else_3
   if − then − else_4
  od
 od
od
```

Fig. 5. The same functionality as in Figure 4, but the middle loop is unrolled

There are just not enough identical operations on independent data. Therefore we must attempt to increase the SIMD-opportunities present in the code.

3.1 Code Transformations

In a normal image reconstruction, the radiological path of millions of rays at various angles will be calculated. To increase the amount of available parallelism, we decided to process four such parallel rays in lock-step. This can be done because the calculations of the different rays do not interfere. It is equivalent to loop unrolling and software pipelining the loop over four parallel rays. This transformation is shown in Figures 4 and 5. It leaves us with one inner loop, in which each of the four rays advances one pixel per iteration. The control flow graph (CFG) of this inner loop is shown in Figure 6. Note that the `then`- and `else`-blocks represent only four operations (eq. 23) and are therefore still not suitable for exploiting subword parallelism.

To concentrate most operations in one basic block, we transformed the

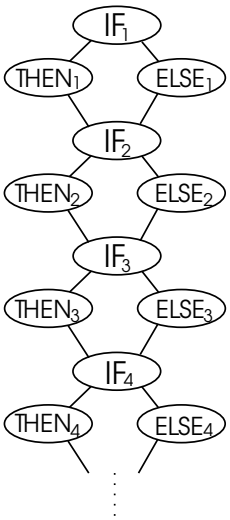

Fig. 6. The code of four parallel rays is executed in lock-step. The first `if`-test decides which step the first ray will take, the second `if`-test, which step the second ray will take, etc.

original CFG into the one depicted in Figure 7. The `if`-tests are hoisted, and as a result, the basic blocks at the right of Figure 7 each perform 1 of the 16 combinations of steps of the 4 rays. In these basic blocks, there are now four times four operations present. This means we finally have basic blocks with enough independent but identical operations to start considering SIMD instructions to speed up the calculations.

Our transformations have an extra benefit; they already result in a major extra speedup due to improved cache behavior. One can verify in Figure 8 that by processing the four rays in lock-step the spatial locality of the data is considerably improved.

In general, the proposed transformations can be applied to any loop with independent operations. In the incremental algorithm this loop corresponds to the loop over parallel rays.

3.2 Data Transformations

So far, the algorithm uses floating-point numbers to represent the densities of the image and distances during image traversal. The VISual instruction set [3,8] on the UltraSPARC, like most architectural multimedia extensions, on the other hand operates on fixed-point data. We found that we could fit the densities of the pixels in bytes and the distances in halfwords (16-bit) without losing much quality in the resulting image. To achieve this, we had to insert code before and after the inner loops that converts floating-point to fixed-point and vice versa.

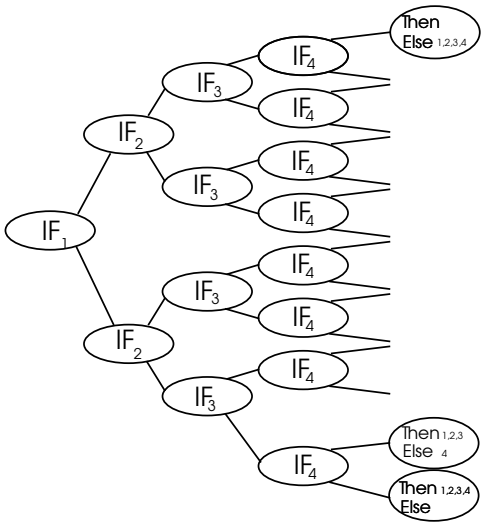

Fig. 7. The CFG of Figure 6 is transformed by performing all the `if`-tests in advance and creating 16 basic block that do the calculations for each of the 16 outcomes of the tests. An oval which contains the notation `then 1,2` and `else 3,4` represents the code equivalent to the `then`-blocks of ray 1 and 2 and the `else`-blocks of ray 3 and 4.

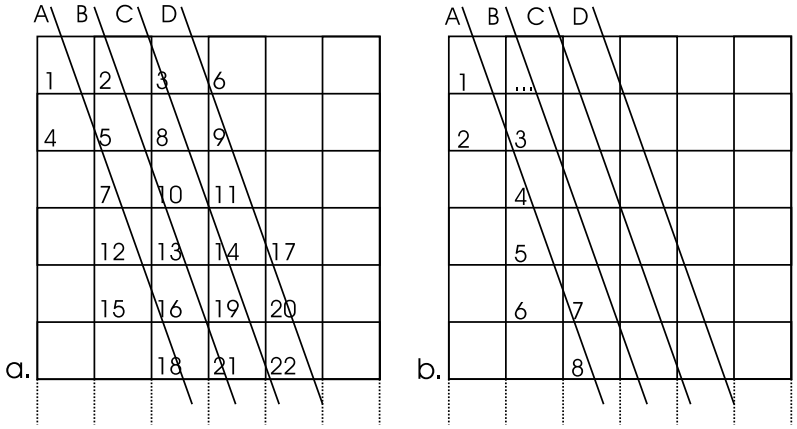

Fig. 8. In part a, 4 rays are calculated in lock-step. The figures indicate the order in which pixels are accessed for the first time. Most pixel values will be in the cache before that first access, due to previously accessed pixels. Part b shows the same figures, without loop unfolding of the code. When ray A is finished, the pixels that once were loaded into the cache will no longer be present when B needs them.

It was impossible to let the whole program work with fixed-point subwords. One reason is that the initializing code before the inner loop and calculations outside the outer loop of the code in Figures 4 and 5 cannot be done on fixed-point subwords. A second reason is that during one execution of the outer loop, the data ranges of bytes and halfwords are sufficient, but the ranges needed in different executions of the outer loop differ so much that we need to scale the data between each such execution.

While the conversion of the different $\alpha's$ between floating-point and fixed-

point representations takes place before and after each inner loop, and the representation of the densities of the image is converted before and after the outer loop in each approximation step, these conversions have no detrimental effect on the total execution time.

3.3 Subword Parallelism

Finally, the algorithm is suitable for recoding the inner loops using subword parallelism. This is done manually, since the compiler at this moment does not support VIS-instructions. A method with inline assembler templates as described in [11,3] did not prove successful either. Luckily, only the bottom basic blocks of Figure 7 have to be reprogrammed this way. Because they all are very alike, this poses no problem.

The VISual instruction sets provides all the necessary instructions, such as parallel comparisons, additions and multiplications. We could for example replace the four successive `if`-tests with one parallel comparison and a jump-table that is indexed with the result of the parallel comparison.

Some other architectures do not have all the necessary instructions. The SIMD-instructions in the TriMedia processor core [5] for example have no parallel comparison or multiplication, so another approach has to be found there.

4 Results

Both the original algorithm of Siddon and our incremental versions (sequential and lock-step) were implemented in C, using the SCO 4.0 optimizing compiler for UltraSPARC-II. We performed measurements on a dual 167 MHz UltraSPARC-II-processor from Sun with 512 Kb cache for each processor and 128 MB of RAM. Both implementations use doubles for floating-point calculations. No manual optimizations were performed.

We embedded our incremental algorithm and Siddon's in an existing 2D PET reconstruction program, *emvox2d*. This program is developed by the Medical Image Processing Group (MIPG), Department of Radiology, University of Pennsylvania, Philadelphia, Pennsylvania, USA. It uses the ML-EM algorithm, in which radiological path calculation takes about 90% of the time.

The results are shown in Figure 9. Execution time scales with the linear dimensions of the reconstructed image. Any non-linearities arise from cache behavior. Pentium and PPC604 C-implementations result in about the same speedup,

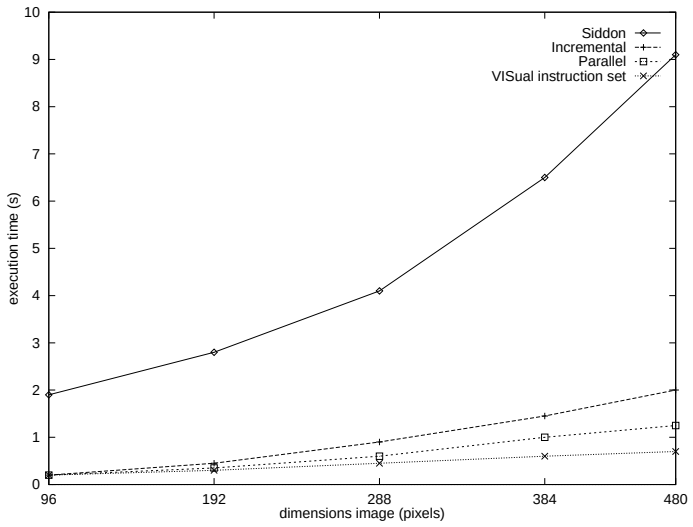

Fig. 9. Execution time against side dimensions (pixel) of the image.

except for the loop unrolling optimization on Pentiums: these processors do not have enough registers to profit from loop unrolling. For large images we achieved a 13-fold speedup. This can be factored in a 4.5-fold speedup from the incremental algorithm, another 1.6-fold speedup by loop unrolling the C-code and a final 1.7-fold performance gain by using the VISual instruction set. The latter is due to more efficient instruction use (SIMD) and better cache behavior, because of the smaller data format used.

In 3D image reconstruction, two types of reconstruction exist. In the first type a 3D image consists of several 2D planes. For each plane, a 2D reconstruction algorithm is used, so the same speed-up is expected. The second type is a full 3D image reconstruction in which voxels are used instead of pixels. For this approach we expect about the same speed-up, but further measurements are necessary.

5 Conclusion

We have proposed an incremental algorithm for the calculation of the radiological path, which improves significantly on the performance of the Siddon algorithm but maintains accuracy. By several transformations we were able to improve cache performance and could eventually implement the algorithm in the VISual instruction set on an UltraSPARC. In doing so we achieved a speedup by a factor of 13.

Acknowledgment

Filip Jacobs introduced us in this matter. We also thank Erik Sundermann for his useful tips in writing this article. Koen Denecker performed some extra measurements on numerous machines, which is greatly appreciated.

References

[1] Gerry Kane. *PA-RISC 2.0 architecture*. Hewlett-Packard professional books. PTR Prentice Hall, Upper Saddle River, New Jersey 07458, USA, 1996.

[2] SPARC International, Inc. *The SPARC Architecture Manual, version 9*. PTR Prentice Hall, 113 Sylvan Avenue, Englewood Cliffs, New Jersey 07632, USA, 1994.

[3] Sun Microelectronics. *Visual Instruction Set (VIS) User's Guide*, 1.0 edition, 1996 April.

[4] Sun Microsystems, Inc. Business. *UltraSPARC Programmer Reference Manual*, 1.1 edition, September 1995.

[5] TriMedia Division Systems Software Group. *TriMedia databook*. Philips Semiconductors, 1.0 edition, November 1996.

[6] L. A. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Transactions on Medical Imaging*, 1(2):113–122, October 1982.

[7] Robert L. Siddon. Fast calculation of the exact radiological path for a three-dimensional CT array. *Medical Physics*, 12(2):252–255, March 1985.

[8] Marc Tremblay, J. Michael O'Conner, Venkatesh Narayanan, and Liang He. VIS speeds new media processing. *IEEE Micro*, 16(4):10–20, August 1996.

[9] David Davidian. VIS in RAID controllers, March 1997. `http://www.sun.com/sparc/vis/RAID.html`.

[10] Filip Jacobs. *Statistische reconstructiemethoden voor 3D Positron Emissie Tomografie*. Aanvraagdossier voor een tweede termijn van een IWT-specialisatiebeurs, Universiteit Gent, April 1996.

[11] Daniel S. Rice. High-performance image processing using special-purpose CPU instructions: The UltraSPARC visual instruction set. Technical report, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, March 1996. `http://cs-tr.cs.berkeley.edu`.

About The Authors

Mark Christiaens was born in Tielt, Belgium, in 1974. He graduated in computer science and engineering at the University of Gent, Belgium, in 1997. As a Ph.D. student, he is currently researching the debugging of distributed programs at the Electronics and Information Systems Department of the Faculty of Applied Sciences at the University of Gent.

Bjorn De Sutter was born in Gent, Flanders, in 1974. He graduated in computer science and engineering at the University of Gent, Flanders, in 1997. He is currently, as a Ph.D. student, researching whole-program optimization at link-time at the Electronics and Information Systems Department of the Faculty of Applied Sciences at the University of Gent.

Koen De Bosschere was born in Oudenaarde, Belgium, in 1963. In 1986 and 1987 respectively, he graduated in electronic engineering and computer science at the University of Gent. In 1992, he obtained his doctoral degree at the same university. He is now research associate with the Fund for Scientific Research - Flanders, and lecturer at the ELIS Department of the University of Gent.

Jan Van Campenhout was born in Vilvoorde, Belgium, on August 9, 1949. He received a Degree in Electro-Mechanical Engineering from the University of Gent, in 1972; and the MSEE and Ph.D. Degrees from Stanford University, in 1975 and 1978, respectively. Prof. Van Campenhout teaches courses at the Faculty of Applied Sciences of the University of Gent, Belgium and is head of the ELIS department.

Ignace Lemahieu was born in Varsenare, Belgium, in 1961. He graduated in physics from the University of Gent in 1983, and obtained his doctoral degree in physics in 1988 from the same university. He joined the department of Electronics and Information Systems (ELIS) at the University of Gent in 1989 as a Research Associate with the National Fund for Scientific Research (N.F.W.O.), Belgium. He is now a professor of Medical Image Processing and head of the MEDISIP research group.