Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik • Lehrstuhl für Informatik 10
Prof. Dr. Ulrich Rüde

Technical Report 06-5

Performance results for optical flow on an Opteron cluster using a parallel 2D/3D multigrid solver

H. Köstler, F. Deserno and C. Möller

May 24, 2006

Lehrstuhl für Informatik 10 (Systemsimulation)
Cauerstraße 6
D-91058 Erlangen
Tel.: +49 9131 85 28923
Fax: +49 9131 85 28928
http://www10.informatik.uni-erlangen.de

This document was set using LaTeX.

Optical flow is a widely used method in machine vision to compute an approximation to real motion in a sequence of images. Using a variational model for the optical flow leads to a coupled system of PDEs that has to be solved efficiently. Therefore, we implemented a multigrid solver based on the Galerkin approach. In order to achieve realtime performance, parallelisation of the code is a necessity. In this report we present several performance results on an Opteron cluster on natural and synthetic images both in 2D and 3D.

Contents

1 Introduction

Optical flow is commonly defined to be the motion of brightness patterns in a sequence of images. Usually it has to be computed in realtime e.g. for applications in robotics. In addition, recent interest has been shown to extend the study of the optical flow from 2D plane images to 3D volume data sets, where it is related to image registration (e.g. for CT images [1], PET volumes [2] and MR images [3]).

Differential approaches, which estimate velocity vectors from spatial and temporal intensity derivatives are most used since they allow for a more efficient exploration of the solution space and result in lower complexity and better accuracy. They are based on the brightness conservation assumption which leads to an ill-posed problem that is often solved by imposing additional assumptions that are incorporated into the original problem via regularisation techniques. A standard model, which is due to Horn and Schunck [4], results in a system of elliptic PDEs of reaction diffusion type. The second order terms are induced by the regularisation and become straight forward Laplace operators. We call this type of regulariser constant diffusion. The zero order terms are linear (but variable) and are computed from the derivatives of the image data. Since the image data (and even more so its derivatives) is usually nonsmooth nontrivial problems arise. This becomes even more important, if one extends the Horn-Schunk model by using a different, more advanced regulariser [5].

Our goal was to make existing fast multigrid based solvers for optical flow (e.g. [6], [7], [8]) more robust and extend them to more advanced regularisers. In [9] it was shown that for constant diffusion using a variational multigrid based on Galerkin approach leads to a good and stable performance on all sort of images in 2D. We extended the approach to 3D [10] and started to parallelise it [11], [12]. Another efficient way of solving the optical flow problem using multigrid is presented in [6].

In this report we present scalability results for the optical flow problem on an Opteron cluster in 2D and 3D.

The outline of the paper is as follows. First, we present in Sec. 2 the optical flow problem using an image-driven isotropic diffusion based regulariser. We then summarise the parallel multigrid scheme for this model and give some details on the implementation. In Sec. 3, we describe the hardware used and show some performance results.

2 Optical Flow Problem

2.1 Optical Flow Model using a image-driven isotropic regulariser

In the following, we summarise the optical flow problem in 2D. The extension to 3D can be found in [12]. Assume we are given a sequence of 2D images with an intensity value $I(x, y, t)$ of the image point (x, y) at time t. The intensity values $I(x, y, t)_{x,y,t}$ are supposed to be described by a differentiable function $I : \Omega \times [0, T] \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$ is the image plain and T is a strictly positive scalar describing the final time. The partial derivatives of I in the direction of x, y and t are denoted by I_x, I_y and I_t, respectively. The underlying optical flow assumption says, that image objects keep the same intensity value under motion for at least a short period of time. In term of equations, this can be stated as follows: $\forall (x, y) \in \Omega, \forall t \in [0, T]$,

$$I(x, y, t) = I(x + dx, y + dy, t + dt).$$

Using Taylor expansion and dropping the nonlinear terms, we get the 2D optical flow constraint equation (OFCE):

$$I_x u + I_y v + I_t = 0,$$

where $(u, v) = (\frac{dx}{dt}, \frac{dy}{dt})$ is the 2D velocity vector. This equation defines an ill-posed problem which is solved via regularisation [13]. Therefore in the simplest case the optical flow is supposed to have smooth variations in the sense that neighboring points have almost the same velocity. The (OFCE) is hence replaced by the energy functional:

$$E(u, v) = \int_x \int_y \left[(I_x u + I_y v + I_t)^2 + \alpha (|\nabla u|^2 + |\nabla v|^2) \right] \mathrm{d}x \, \mathrm{d}y \tag{2.1}$$

that has to be miminised. The positive function $\alpha : \mathbf{R} \to \mathbf{R}_+$ is used for adjustment between the data and the additional smoothness constraint and is small at locations where the magnitude of the spatial image gradient is large in order to preserve discontinuities. This form of regularisation is called image-driven isotropic diffusion.

The Euler-Lagrange equations derived from the minimisation problem (2.1) define a system of elliptic PDEs with nonconstant coefficients depending on the image data for the zero-order terms:

$$\begin{aligned} \nabla \alpha \nabla u - I_x (I_x u + I_y v + I_t) &= 0 \\ \nabla \alpha \nabla v - I_y (I_x u + I_y v + I_t) &= 0. \end{aligned} \tag{2.2}$$

This system is symmetric with respect to the components of the velocity u and v.

2.2 Multigrid Scheme

We solve (2.2) it using an efficient multigrid solver. Multigrid methods are known to be among the most efficient solution methods for elliptic PDEs. For a comprehensive overview on multigrid methods, we refer to [14] and [15]. The core idea of multigrid is to use a sequence of coarse grids as a means to accelerate the solution process (by relaxation such as Gauss-Seidel) on the finest grid. This leads to recursive algorithms like the so-called V- or W-cycle, which traverse between fine and coarse grids in the mesh hierarchy. Since ultimately only a small number of relaxation steps on each level must be performed, multigrid provides an asymptotically optimal method whose complexity is only $\mathcal{O}(N)$, where N is the number of mesh points.

The first attempts to use multilevel techniques for low-level problems in computer vision, and particularly for the optical flow are due to Glazer [8] and Terzopoulos [16]. They both reported improvement in performance but only by testing simple synthetic images. Enkelmann [7] adopted a coarse-to-fine strategy and used an image pyramid to subsample the images into different resolutions. The spatial intensity gradient information need not to be transferred to the current coarse level as in [8] since they could be calculated from the corresponding level of the image pyramid. Experiences on real 2D images have shown that both methods (direct restriction of the image gradient and building an image pyramid) lead to similar results when used in a standard multigrid: acceptable convergence rate for *smooth* images and slow convergence or even divergence for highly-textured images, especially if we consider more than two levels in the multigrid hierarchy. Clearly, there is a loss of information when we represent the problem on coarse levels. Battiti et al [17] stated that a usual multigrid cycle is not appropriate due to a possible information conflict between different scales, and they preferred rather a one-way (coarse-to-fine) strategy. However, one-way multigrid methods – also known as cascadic multigrid – do not reach the optimal efficiency of a classical (bidirectional) multigrid method with a good coarse grid approximation of the original problem. A close look on system (2.2) shows that the finest grid operator has some nice properties that could be exploited for coarse grid approximation; we present the 2D case that can also be found in [9] and refer for the 3D case to [12].

First, we write system (2.2) as follows

$$L\xi = F \tag{2.3}$$

where

$$\xi = \begin{pmatrix} u \\ v \end{pmatrix} \quad , \quad F = \begin{pmatrix} -I_x I_t \\ -I_y I_t \end{pmatrix},$$
$$L = L_d + L_r \quad , \quad L_r = \begin{pmatrix} -\nabla\alpha\nabla & 0 \\ 0 & -\nabla\alpha\nabla \end{pmatrix},$$

and L_d is a 2×2 block diagonal matrix with entries

$$\begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

We also use the notation

$$L = \begin{pmatrix} L_{xx} & L_{xy} \\ L_{yx} & L_{yy} \end{pmatrix} = \begin{pmatrix} -\nabla\alpha\nabla + I_x^2 & I_x I_y \\ I_x I_y & -\nabla\alpha\nabla + I_y^2 \end{pmatrix} .$$

It is clear that L_r is symmetric positive definite. Moreover, one can show that L_d is symmetric positive semi-definite, too. Hence the sum L is also symmetric positive definite. This suggests to put the linear system (2.3) into its equivalent variational minimisation form

$$\xi = arg \min_{\Omega} a(\eta).$$

Here a is the quadratic form given by $a(\eta) = \frac{1}{2}(L\eta, \eta) - (F, \eta)$.

Let's denote by h and H the current resolution and the next coarser resolution, respectively. Let also $I_H^h : \Omega^H \mapsto \Omega^h$ be a full rank linear mapping. An optimal coarse grid correction $I_H^h \xi_H$ of the current approximation ξ_h is characterised by

$$((I_H^h)^T L_h I_H^h)\xi_H = (I_H^h)^T (F - L_h \xi_h) .$$

The Galerkin approach consists then to choose the coarse grid operator as follows

$$L_H = I_h^H L_h I_H^h \quad and \quad I_h^H = (I_H^h)^T.$$

For our system, this yields

$$\begin{aligned} L_H &= \begin{pmatrix} R & 0 \\ 0 & R \end{pmatrix} \begin{pmatrix} L_{xx,h} & L_{xy,h} \\ L_{yx,h} & L_{yy,h} \end{pmatrix} \begin{pmatrix} P & 0 \\ 0 & P \end{pmatrix} \\ &= \begin{pmatrix} R\,L_{xx,h}\,P & R\,L_{xy,h}\,P \\ R\,L_{yx,h}\,P & R\,L_{yy,h}\,P \end{pmatrix} \end{aligned}$$

where R and P are any 2D-grid transfer operators satisfying $P = R^T$. In our implementation, R is the full weighting and P is the bilinear interpolation, see [15]. The components of our V-cycle are described as follows

- Vertex-centered grid and standard coarsening
- Coupled lexicographic point or line Gauss-Seidel smoother
- Full weighting and bilinear interpolation
- Galerkin coarse grid approximation (GCA approach)

2.3 Implementation and Parallelisation Issues

In our implementation, all images are presmoothed with a Gaussian kernel before applying convolution masks to compute the derivatives. For the discretization of the PDEs and the image derivatives we used finite differences. The spatial image derivatives are

calculated using the stencil mask $\frac{1}{12}[1 \quad -8 \quad 0 \quad 8 \quad -1]$, the temporal derivatives using the stencil mask $[-1 \quad 1]$.

Again details for the 3D discretization can be found in [12]. The code is written in C++ and parallelised using MPI ([18], [19]). For the parallel implementation of the multigrid V-cycle we apply a grid partitioning strategy rather than domain decomposition since it is known to keep the asymptotic convergence behavior of the sequential algorithm [20]. The image domain is split up into parts along all directions and a layer of ghost cells on each boundary is introduced to store the outermost values of the neighboring processor. Due to the dependencies of the unknowns in the parallel smoother, we use a red-black Gauss-Seidel solver instead of a lexicographic. After the update of each color in a Gauss-Seidel step, the corresponding values of ghost cells are exchanged between neighboring processors. In addition to the data exchange in the smoother, we have to exchange the computed residual and the corrected solution on each level in the V-cycle (see Algorithm 1). Especially for a larger number of coarse levels, the amount of communication needed for parallel multigrid becomes very large. A detailed discussion of the parallelisation of multigrid is e.g. found in [15].

Algorithm 1 – Parallel V-cycle

```
1: Presmoothing {exchange ghost cells of each color}
2: Compute residual
3: Restrict residual
4: Exchange residual
5:
6: if coarsest level then
7:    smooth several times {exchange ghost cells of each color}
8: else
9:    call V-cycle recursively on next coarser level
10: end if
11:
12: Correct current solution with interpolated solution from coarser level
13: Exchange solution
14: Postsmoothing {exchange ghost cells of each color}
```

The full scheme is described in the following.

1. Main processor reads frames
2. Mesh partitioning and the main processor distributes the needed parts of the frames to the other processors
3. Each processor computes spatial and temporal derivatives
4. Each processor calculates the finest grid operator and the right hand side (RHS)
5. Each processor sets up the coarse grid operators hierarchy using the Galerkin scheme

6. Each processor participates in the multigrid V-cycling
7. The results of each processor are collected by the main processor
8. Main processor writes out the optical flow field

3 Experimental results

3.1 Performance metrics

In order to assess the quality of the implementation, performance metrics for serial and parallel benchmarks have to be introduced.

3.1.1 Speed-Up

As a measure for single processor performance we will use CPU-time t. In order to get an impression of the parallel performance, we define the speed-up s as the fraction of the run time on 1 CPU compared to the run time on n CPUs. However, the definition of s has to be done more precisely [21].

Run times depend on the number of CPUs n and the given problem size N and can be devided into two parts – the serial t_s and parallel execution time t_p.

$$t = t(n, N) = t_\mathrm{s} + t_\mathrm{p} \equiv 1$$

The sum of both parts is defined as one. Speed-up s can be written as

$$s = \frac{t(1, n^x N)}{t(n, n^x N)} \tag{3.1}$$

which is the fraction of the serial run time for a parallel problem compared to the run time in parallel for the same problem[1]. Parallel problem in this sense means that the problem size N is scaled by a factor of n^x while $x \geq 0$ (no down scaling). A value of $x = 1$ means linear scaling $x = 0$ denotes no scaling of the problem size. To make things a bit easier, we assume that only the parallel part of the program depends on the problem size (constant setup times), i.e.

$$t_\mathrm{s} = \text{const.}$$

If we increase CPU count to n, the time for the parallel part decreases by a factor of $1/n$. It increases by n^x since the problem size is scaled simultaneously. Using Eq. 3.1, the speed-up s can now be rewritten as

$$s = \frac{t_\mathrm{s} + t_\mathrm{p} n^x}{t_\mathrm{s} + \frac{t_\mathrm{p}}{n} n^x} = \frac{t_\mathrm{s} + t_\mathrm{p} n^x}{t_\mathrm{s} + t_\mathrm{p} n^{x-1}} \tag{3.2}$$

[1] In practice, it might become difficult to determine the serial run time since the amount of memory is limited per processor. As an approximation, one can use $t(1, n^x N) \approx t(1, N) \cdot n^x$.

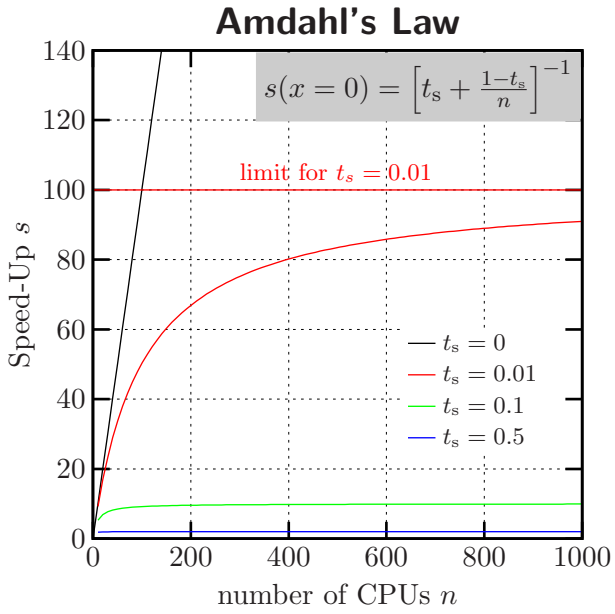


Figure 3.1: Amdahls Law. $t_s > 0$ results in an upper limit for speed-up s. This is also called strong scaling since the problem size remains constant.

Considering different values for x will lead us to two important cases.

Amdahl's Law – Strong Scaling

The first obvious possibility is $x = 0$ which means that we do not scale the problem size at all. In this case Eq. (3.2) becomes

$$s(x=0) = \frac{t(1,N)}{t(n,N)} = \frac{t_s + t_p}{t_s + \frac{t_p}{n}} = \frac{1}{t_s + \frac{1-t_s}{n}} \tag{3.3}$$

which is also called Amdahls Law or strong scaling. Strong means that communication more and more prevails with an increasing number of CPUs so that it is hard to maintain the speed-up. Figure 3.1 shows Amdahls Law for three different cases of t_s. For $t_s = 0$ we receive ideal speed-up since there is only the fully parallelisable part left. Every value large than 0 results in an upper limtit for s, even for an infinite number of CPUs.

$$\begin{aligned} s(x=0, t_s=0) &= n \\ s(x=0, n\to\infty) &= 1/t_s \end{aligned}$$

Gustafson's Law – Weak Scaling

Now we scale the problem size with the number of processors by a factor of n^x, $x > 0$. This is called weak scaling since the problem size per CPU is increased with the CPU

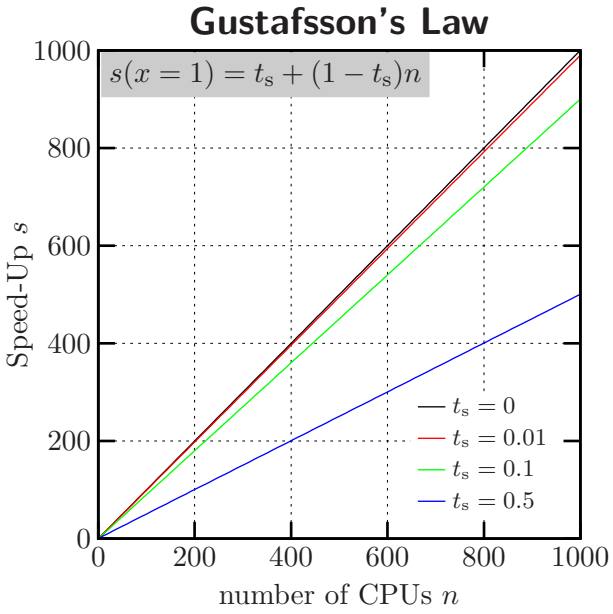


Figure 3.2: Gustafson's Law. In contrast to Amdahl we now have no upper limit to the speed-up. t_s only determines the slope of the speed-up.

count. One can derive three different cases from Eq. (3.2):

$$\begin{aligned}
s(0<x<1) &= \frac{t_s + t_p n^x}{t_s + \frac{t_p}{n^{1-x}}} & \overset{n\to\infty}{\longrightarrow} \quad \frac{t_p n^x}{t_s} &= \frac{1-t_s}{t_s} n^x \\
s(x=1) &= \frac{t(1,nN)}{t(n,nN)} = t_s + t_p n & \overset{n\to\infty}{\longrightarrow} \quad t_p n &= (1-t_s)n \\
s(x>1) &= \frac{t_s + t_p n^x}{t_s + t_p n^{x-1}} & \overset{n\to\infty}{\longrightarrow} \quad \frac{t_p n^x}{t_p n^{x-1}} &= n
\end{aligned}$$

The second case

$$s(x=1) \overset{n\to\infty}{\longrightarrow} (1-t_s)n \tag{3.4}$$

is called Gustafson's Law. Obviously, there is no upper limit for s like in Amdahl's Law (see Fig. 3.2) so that an infinite speed-up is theoretically possible. However, the larger t_s the smaller will be the increase in speed-up for a higher CPU count.

3.1.2 Parallel efficiency

Parallel efficiency e can be defined as the run time on 1 CPU compared to n times the run time on n CPUs – or simply the speed-up devided by n:

$$e = \frac{t_s + t_p n^x}{n\,(t_s + t_p n^{x-1})} = \frac{s}{n}. \tag{3.5}$$

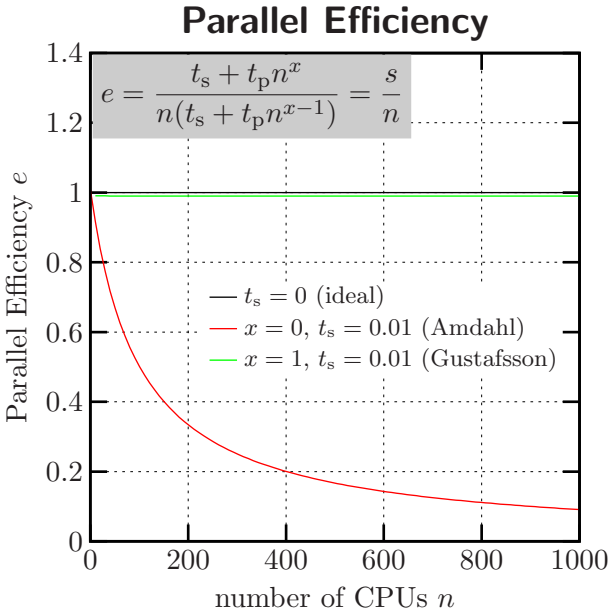


Figure 3.3: Parallel Efficiency. If the problem size is not scaled, e goes to 0 for large CPU counts (Amdahl). In the opposite case, e approaches a constant value depending on t_s.

Table 3.1: Relevant times in practice for speed-up and efficiency measurement in Amdahl's and Gustafson's case.

	Amdahl	**Gustafson**
speed-up s	$\frac{t(1,N)}{t(n,N)}$	$\frac{t(1,nN)}{t(n,nN)} \approx \frac{nt(1,N)}{t(n,nN)}$
efficiency e	$\frac{t(1,N)}{nt(n,N)}$	$\frac{t(1,nN)}{nt(n,nN)} \approx \frac{t(1,N)}{t(n,nN)}$

We can now have a look at two important cases considering Amdahl's and Gustafson's Law.

$$e(x=0) = \frac{t_s + t_p}{n\left(t_s + \frac{t_p}{n}\right)} = \frac{1}{t_s n + t_p} = \frac{1}{1 + t_s(n-1)} \overset{n\to\infty}{\longrightarrow} \frac{1}{t_s(n-1)}$$

$$e(x=1) = \frac{t_s + t_p n}{n\,(t_s + t_p)} = t_p + \frac{t_s}{n} = 1 - t_s + \frac{t_s}{n} \overset{n\to\infty}{\longrightarrow} 1 - t_s$$

Figure 3.3 shows e for $t_s = 0.01$ and the selected cases. It can be seen that e goes to 0 for large CPU counts in the Amdahl case ($x = 0$), i.e. when the problem size remains constant. For $x = 1$ (Gustafson) the efficiency approaches a constant value for an increasing number of CPUs.

Table 3.1 summarises the relevant times to be measured in practice for speed-up and efficiency.

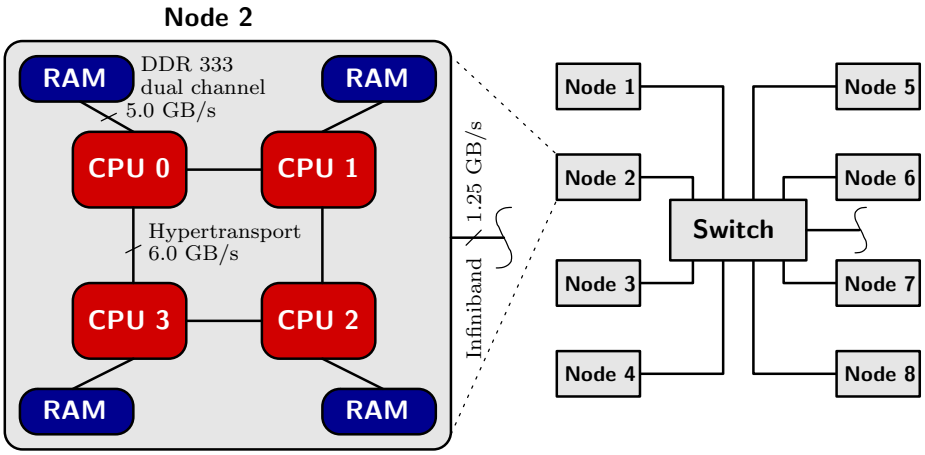


Figure 3.4: Configuration of the benchmark cluster. A compute node contains 4 CPUs, each of which connected to their own local memory, while the CPUs have a Hypertransport interconnect. An Infiniband network provides communication between the quad-nodes.

3.2 Benchmark Architecture

All benchmarks were performed on a cluster of AMD Opteron CPUs. The cluster consists of 4-way nodes, each containing four Opteron 248 CPUs running on 2.2 GHz with an L2-cache of 1 MByte. A 4-way node has 16 GByte of main memory and is connected to all other nodes by an Infiniband network. The latter provides a bandwidth of 1.25 GByte/s. The whole cluster has a theoretical peak performance of

$$\overbrace{8}^{\text{number of nodes}} \cdot \underbrace{4}_{\text{CPUs per node}} \cdot \overbrace{4.4\ \text{GFlop/s}}^{\text{peak for 1 CPU}} \approx 140\ \text{GFlop/s}$$

and a total memory of 128 GByte. Figure 3.4 shows the setup of a node. Within a node, CPUs communicate using Hypertransport technology for a bandwith of up to 6.0 GByte/s. Each CPU manages 4 GByte of RAM which is also accessible by all other CPUs within the node. The connection from CPU to its local memory delivers up to 5.0 GByte/s via DDR333.

3.3 Results

We tested our parallel multigrid implementation using Galerkin discretisation on 2D and 3D data sets of different sizes. For the sake of simplicity $I_x = I_y = I_z = I_t = 1$ was used. Performance in terms of the run time of one V-cycle was measured.

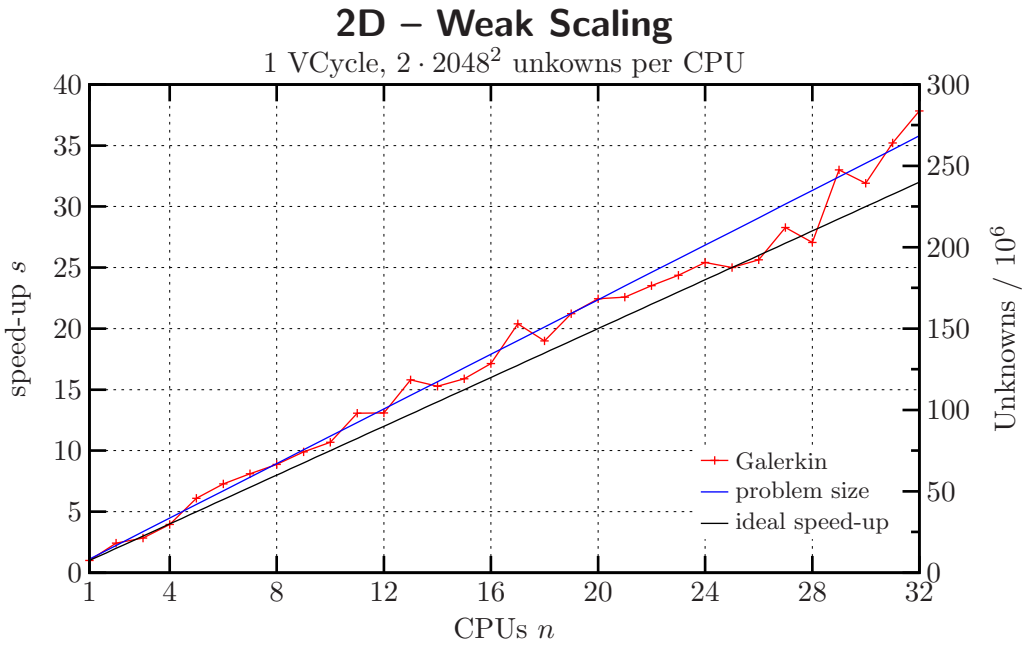


Figure 3.5: Weak Scaling (Gustafson's case) in 2D. s is close to the ideal speed-up.

3.3.1 2D

In the 2D case $2 \cdot 2048^2 \approx 8.4 \cdot 10^6$ unknowns per CPU were used. When using weak scaling, the number of unknowns is scaled linearly with CPU count. Figure 3.5 shows speed-up s for up to 32 processors. s is very close to the ideal speed-up up to 28 CPUs. Beyond, it even tends to be super linear.

In contrast to weak scaling the number of unknowns in Fig. 3.6 is kept constant (strong scaling). Here we have very similar behavior of s. As expceted, in 2D no impact of communication can be observed.

Figure 3.7 shows parallel efficiency for weak and strong scaling. The slightly super linear speed-up in both cases results in an efficiency larger than 1.

3.3.2 3D

In 3D the number of unknowns per processor was set to $2 \cdot 158^3 \approx 7.9 \cdot 10^6$. Disregarding the relatively large variation in run times, weak scaling results in an almost linear speed-up (Fig. 3.8). Several repetitions lead us to the impression that those variations depend largely on the machines utilisation at a whole. However, the parallel efficiency is now slighlty below one. It is hard to say whether this effect comes from a large communication effort or is due to a high variation in the measurement.

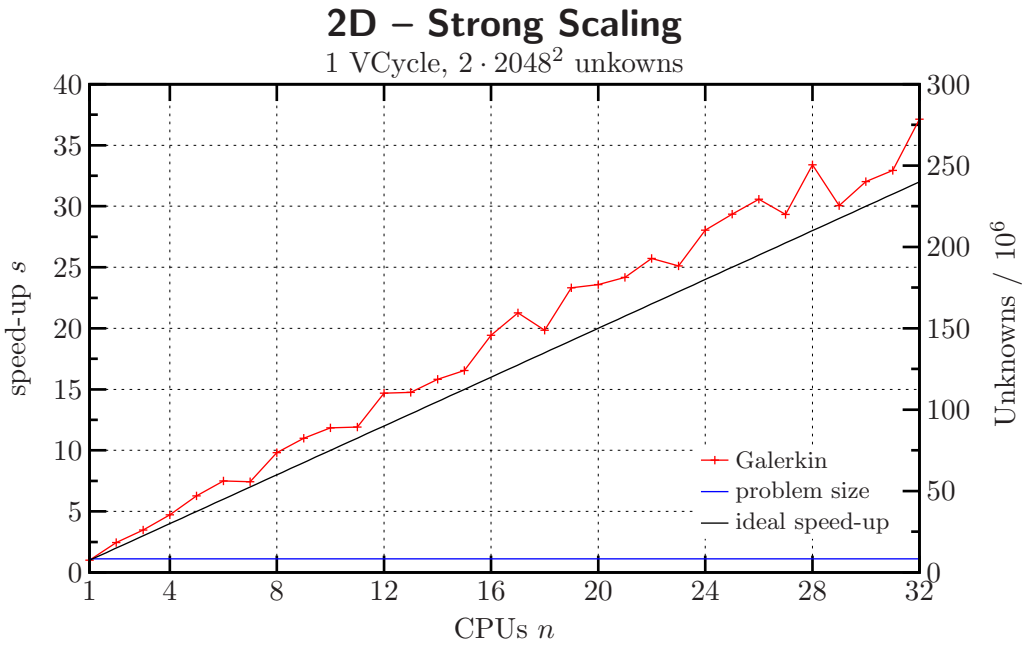


Figure 3.6: Strong Scaling (Amdahl's case) in 2D.

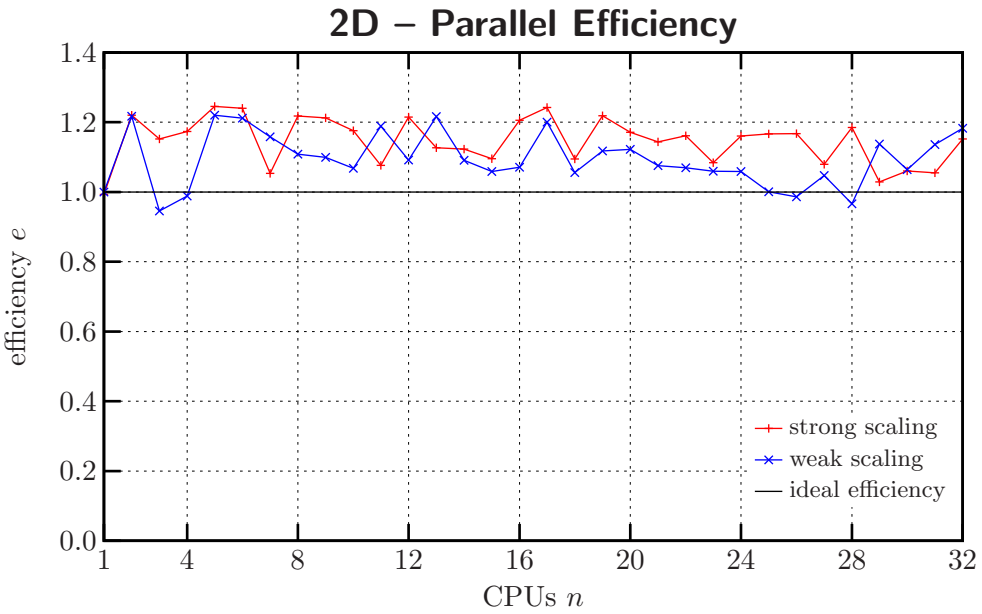


Figure 3.7: Parallel efficiency in 2D.

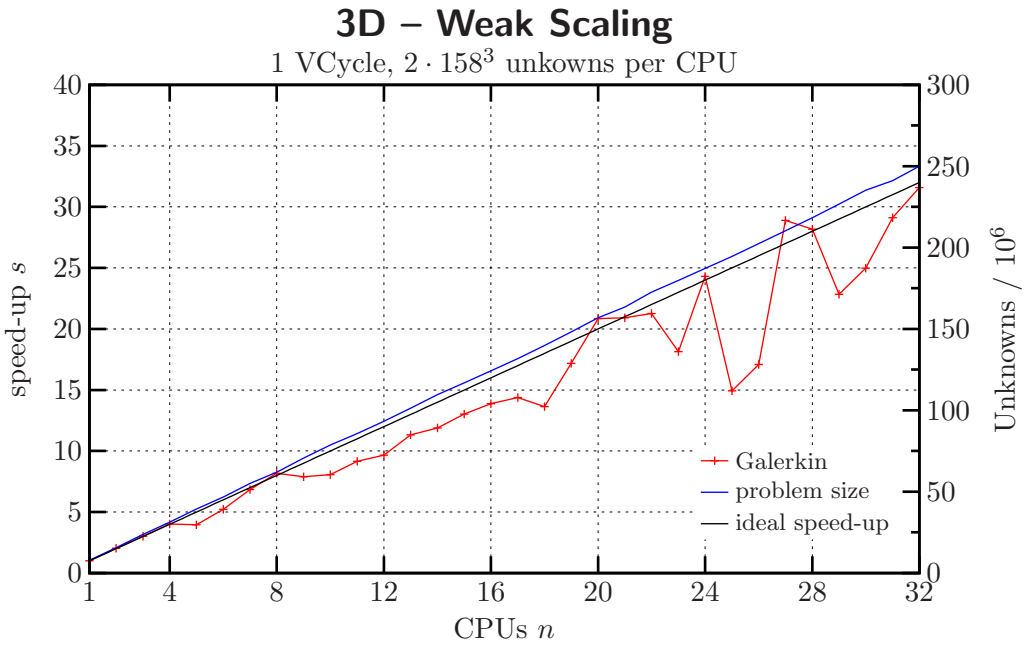


Figure 3.8: Weak Scaling (Gustafson's case) in 3D.

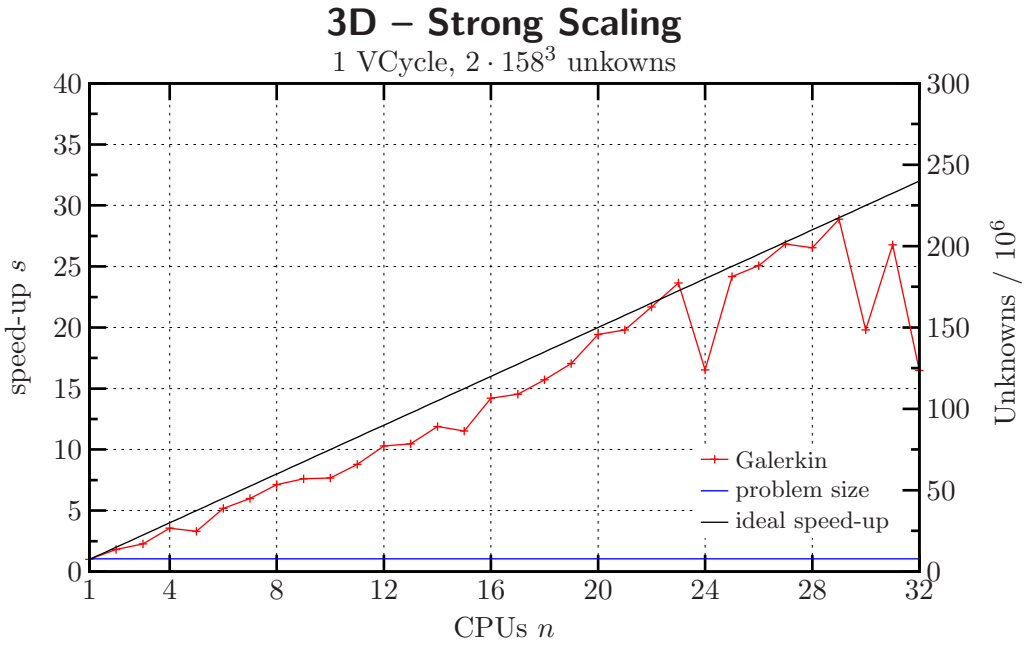


Figure 3.9: Strong Scaling (Amdahl's case) in 3D.

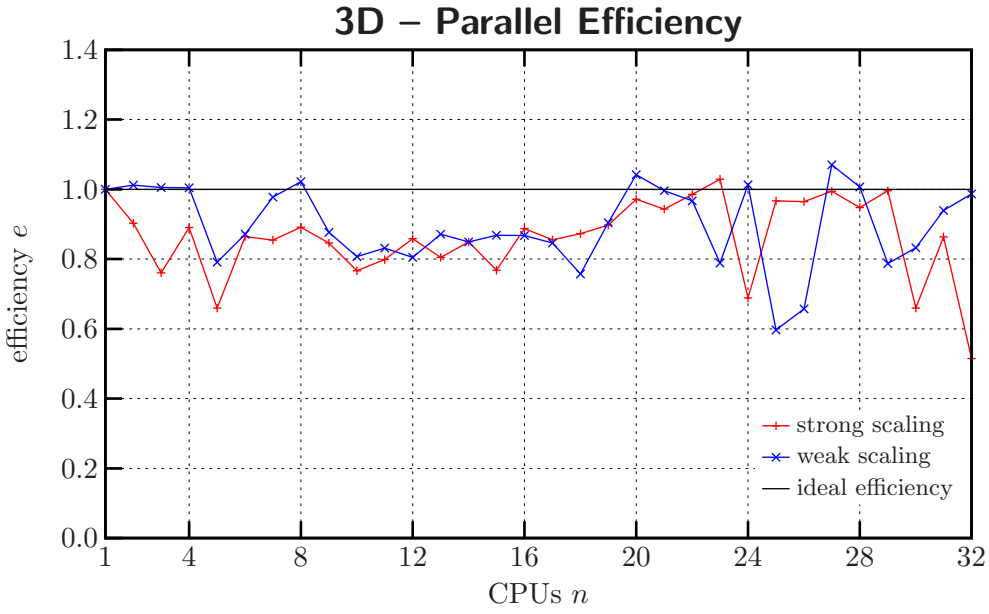
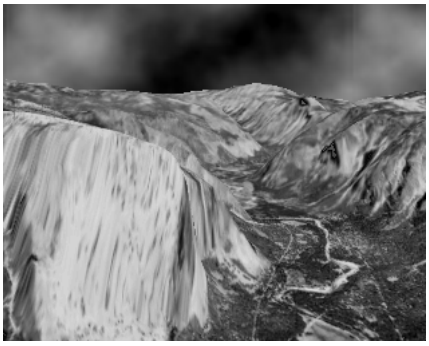


Figure 3.10: Parallel efficiency in 3D.

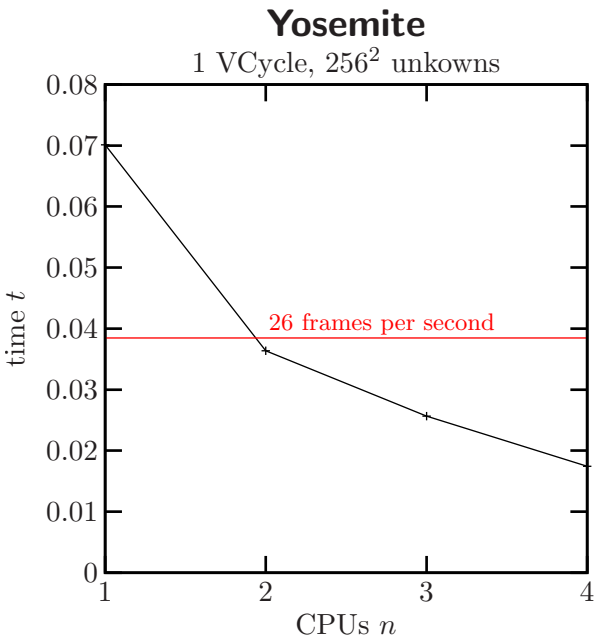
3.3.3 Realtime

In order to present relatime performance we chose the well-known Yosemite sequence (rescaled to 256^2) as an example. Figure 3.11 shows the run time of one V-cylce up to 4 processors using 256^2 unknowns for this application.

Using two CPUs makes it possible to exceed the limit of 26 frames per second.



(a) Yosemite mountain.



(b) Performance on differnt number of CPUs.

Figure 3.11: A typical example for testing the ability for real time computation is the Yosemite sequence (a). Using 2 CPUs, the limit of 26 frames per second can be exceeded (b).

4 Conclusion

We have reported our experience with the multigrid implementation of the 2D and 3D optical flow on parallel machines and have shown the scalability of our parallelisation. The next steps will be the extention to more advanced linear and nonlinear regularisers.

Bibliography

[1] S.M Song and R.M. Leahy. Computation of 3-d velocity fields from 3-d cine ct images of the human heart. *IEEE Transactions on Medical Imaging*, 3(10):295–306, 1991. 1

[2] G.J. Klein and R.H. Huesman. A 3d optical flow approach to addition of deformable pet volumes. *Proceedings of the 1997 IEEE Workshop on Motion of Non-Rigid and Articulated Objects (NAM '97)*, June 1997. 1

[3] N. Hata, A. Nabavi, W.W. Wells, S. Warfield, R. Kikinis, P. Black, and F.A. Jolesz. Three-dimensional optical flow method for measurement of volumetric brain deformation from intraoperative magnetic resonance images. *J. Comput. Assist. Tomogr.*, (24):531–538, 2000. 1

[4] B.K.P Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, (17):185–203, 1981. 1

[5] J. Weickert and C. Schnörr. A theoretical framework for convex regularizers in pde-based computation of image motion. Technical Report 13, University of Mannheim, June 2000. 1

[6] C.Feddern T.Kohlberger A.Bruhn, J.Weickert and C.Schnörr. Variational optical flow computation in real-time. *IEEE Transactions on Image Processing*, 14(5):608–615, 2005. 1

[7] W. Enkelmann. Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing*, (43):150–177, 1988. 1, 2.2

[8] F. Glazer. Multilevel relaxation in low-level computer vision. *Multi-resolution image processing and Analysis (A. Rosenfeld, Ed.) Springer-Verlag*, pages 312–330, 1984. 1, 2.2

[9] E.M Kalmoun and U. Rüde. A variational multigrid for computing the optical flow. *Vision, Modeling and Visualization 2003*, pages 577–584, Berlin 2003. T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, R. Westermann (Eds.). Akademische Verlagsgesellschaft. 1, 2.2

[10] H. Köstler, Christadler I., and U. Rüde. Robust and efficient multigrid techniques for the optical flow problem using different regularizers. Technical Report 05-6, University of Erlangen-Nuremberg, 2005. 1

[11] Ch. Möller. *Paralleler optischer Fluss in 3D*. Studienarbeit, Friedrich-Alexander Universität Erlangen-Nürnberg, 2005. 1

[12] H. Köstler, E.M. Kalmoun, and U. Rüde. Parallel multigrid computation of the 3D optical flow. Technical Report 04-4, University of Erlangen-Nuremberg, 2004. 1, 2.1, 2.2, 2.3

[13] A.N Tikhonov and V.Y. Arsenin. *Solution of ill-posed problems*. Wiley, New York, 1977. 2.1

[14] W. Briggs, V. Henson, and S. McCormick. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, 2000. 2.2

[15] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001. 2.2, 2.2, 2.3

[16] D. Terzopoulos. Image analysis using multigrid methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (8):129–139, 1986. 2.2

[17] R. Battiti, E. Amaldi, and C. Koch. Computing optical flow across multiple scales: an adaptive coarse-to-fine strategy. *International Journal of Computer Vision*, (6):133–145, 1991. 2.2

[18] http://www.mcs.anl.gov/mpi/standard.html. The mpi forum. the mpi message-passing interface standard, 1995. 2.3

[19] W. Gropp. *Using MPI-2*. MIT Press, 1999. 2.3

[20] I. Martín and F. Tirado. Relationships between efficiency and execution time of full multigrid methods on parallel computers. *IEEE Trans. on Parallel and Distributed Systems*, (8):562–573, 1997. 2.3

[21] J. Gustafson. Re-evaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, May 1988. 3.1.1

List of Figures

List of Tables