Accelerated implementation of level set based segmentation

M. J. Piggott[1] P. Vallotton[2] J. A. Taylor[3]
T. P. Bednarz[4]

(Received 31 October 2012; revised 29 May 2013)

Abstract

An Open Computing Language implementation of a level set solver for 2D and 3D image segmentation tasks is presented. An adaptive time stepping algorithm is implemented using an optimised parallel reduction kernel to compensate for a loss of algorithmic parallelisation. For a 2D data set (256×256) the execution is accelerated by a factor of 20 in the adaptive case and 100 in the non-adaptive case compared to a CPU implementation, facilitating real time interactive parameter tuning. For a 3D data set ($384 \times 397 \times 41$) the acceleration factors are 200 and 270 for the adaptive and non-adaptive cases, respectively. Although a single iteration of the adaptive method is slower compared to the non-adaptive scheme, it automatically enforces the Courant–Friedrichs–Lewy condition and reduces the number of user-tuned parameters while safely allowing larger time steps. Open Computing Language optimisations and techniques are discussed.

Subject class: 68U05, 65D18

Keywords: level set method, OpenCL, GPU, segmentation, upwinding, adaptive timestepping

Contents

1 Introduction

The image segmentation problem is to identify objects within images and has become a staple of image analysis and computer graphics literature over the last two decades. Common to all segmentation methods is the need for user interaction in the form of parameter tuning and the specification of initial conditions. With large datasets such an ad hoc approach is time consuming. This prevents real time user interaction with segmented objects, which is an important aspect of modern applications. With the rise of graphics processing units (GPUs) as accelerators for general purpose computing (generally referred to as GPGPU) [15], several researchers sought to address this problem by developing GPU implementations of image segmentation algorithms. These range from applications using graphics primitives [11] to modern approaches utilising NVIDIA's compute unified device architecture (CUDA) [9].

In contrast to earlier work, Open Computing Language (OpenCL) [7] is used for its portability across devices and its cross-vendor support. OpenCL is an open framework for programming heterogeneous computing platforms involving, for example, CPUs, GPUs and digital signal processors, collectively known as OpenCL compute devices. It provides a language based on C99 for writing kernel functions which are executed as work items (threads) on compute devices, and also an application programming interface for managing the platforms, compute queues and memory operations. The focus here is on the use of OpenCL in an environment with a single GPU which executes kernel functions in parallel under a single instruction multiple data (SIMD) architecture, and a 'host' CPU which provides control over memory management, event synchronisation and compute device instruction queueing.

A PDE based level set segmentation algorithm is presented, and an OpenCL implementation of the associated full grid solver is given for both 2D and 3D applications. Important GPU optimisations and techniques are discussed, particularly in relation to a parallel reduction kernel used for implementing an adaptive time stepping algorithm. A performance analysis indicates that on a single Fermi Tesla M2050 GPU interactive speeds are achievable for moderate sized data sets in 2D and 3D for both the adaptive and non-adaptive algorithms, despite the loss of algorithmic parallelism in the adaptive case.

2 Level set segmentation

The level set method [14] is a numerical method for solving multidimensional front and surface propagation problems in an Eulerian framework. In particular, it is used extensively in image processing and image analysis, not only in segmentation applications [10, 11, 12], but also in image denoising and restoration applications [13]. The level set framework was also successfully applied in a diverse range of fields such as computational fluid dynamics and machine learning [18], with a thorough introduction to the field provided by Sethian [16].

To formulate the level set method, consider the time evolution of a closed, smooth $d-1$ dimensional front denoted by $\Gamma(t)$ at time t. For example, if $d=2$ then Γ encodes the evolution of a smooth closed contour in the plane. The front moves along its outward normal vector field with a speed $F_{\Gamma,\Theta}(\mathbf{x},t):\Gamma(t)\times\mathbb{R}^+\to\mathbb{R}$, which may depend on local geometric properties of the front and a set of parameters Θ. The key observation of Osher and Sethian [14] was that one can treat the front as the (zero) level set of a function $\phi:\mathbb{R}^d\times\mathbb{R}^+\to\mathbb{R}$, and simply derive a Hamilton–Jacobi type evolution equation for ϕ. Under general conditions the desired evolution of the front is captured in the evolution of ϕ [6].

The front is defined as

$$\Gamma(t):=\left\{\mathbf{x}\in\mathbb{R}^d:\phi(\mathbf{x},t)=0\right\}. \tag{1}$$

Consider a trajectory $\mathbf{x}(t)$ of a point on the front Γ. Using (1) to compute the total derivative, one obtains

$$\frac{\partial\phi}{\partial t}(\mathbf{x}(t),t)+F_{\Gamma,\Theta}(\mathbf{x}(t),t)\left|\nabla\phi(\mathbf{x}(t),t)\right|=0. \tag{2}$$

To convert (2) to an initial value problem on $\mathbb{R}^d$, the normal speed $F_{\Gamma,\Theta}$ must be extended to $\mathbb{R}^d$ and an initial condition ϕ_0 proposed. The extended function is denoted by F_Θ and satisfies $F_\Theta|_\Gamma=F_{\Gamma,\Theta}$. In this application there is no need to explicitly compute such an extension as the speed function is well defined away from the front.

2.1 Initial conditions

The initial condition ϕ_0 must satisfy $\phi|_{\Gamma(0)} = 0$ and is typically chosen as the signed distance function (SDF) to the initial front:

$$\phi_0(x) = \begin{cases} d\left(x, \Gamma(0)\right) & \text{if } x \text{ is `inside' } \Gamma(0)\,, \\ -d\left(x, \Gamma(0)\right) & \text{if } x \text{ is `outside' } \Gamma(0)\,, \end{cases} \tag{3}$$

where $d\left(x, \Gamma(0)\right) = \min_{p \in \Gamma(0)} \|x - p\|_2$ is the Euclidean point-set distance. To compute ϕ_0, Terriberry's implementation of the SDF algorithm [5] of $O(N)$ complexity was used, where N is the number of pixels in the image. There are two primary technicalities relating to this choice of ϕ which are discussed below.

Firstly, ϕ_0 is not differentiable on the closure of its skeleton $\bar{S}$ [2], that is, the set of points which are equidistant to two or more parts of the front. Thus a weak formulation is required. Under the curvature dependent speed function of Section 2.2, the evolution remains smooth and a physically reasonable unique entropy solution is attained using a Huygen's principle construction [14]. The relevant numerical schemes are discussed in Section 3.

Secondly, ϕ does not remain an SDF as the evolution unfolds since $|\nabla\phi|$ can become unbounded near the zero level set. This was demonstrated for standard Hamilton–Jacobi equations [6], and is well described in the level set literature [13, 16]. That is, the level sets 'bunch up' and numerical instability can arise as spatial derivative computations lose accuracy. To alleviate this, a simple approach was used to extract the front (zero level set of ϕ) periodically, reinitialising ϕ to be the SDF to the current front, and allowing the evolution to continue. Other more sophisticated methods abound, typically based on modifying the level set PDE itself or introducing new PDEs which in effect enforce the property $|\nabla\phi| = 1$ for all t [1, 6, 10].

In terms of segmentation, the initial contour is specified by a user, ϕ_0 computed, and the front is evolved until some implicit stopping criteria, which is built in to the speed function, is fulfilled. Often these stopping criteria are derived from a pre-processed edge-enhanced image, to encourage the front to attain the shape of the boundary [12].

2.2 Speed function

The key element characterising the evolution of ϕ is the speed function F_Θ. For simplicity, the speed function proposed by Lefohn et al. [11] is used, which depends on three scalar parameters $\Theta = \{\alpha, \epsilon, T\}$:

$$F_\Theta(x, t) = \alpha D(x) - (1 - \alpha)\kappa(x, t)\,, \tag{4}$$

where $\kappa(x, t) = \nabla \cdot \left[\nabla\phi(x, t)/|\nabla\phi(x, t)|\right]$ is the mean curvature and $D(x) = \epsilon - |I(x) - T|$ in effect defines a region of interest for the segmentation using the pixel intensity I, an intensity threshold T and a band width ϵ. If $I(x)$ is inside the region to be segmented, defined as an ϵ band around the intensity T, then $D(x) > 0$ and the front expands to enclose x, and vice versa. The parameter $\alpha \in [0, 1]$ controls the trade off between advection dominated and curvature dominated evolution.

In summary, one has the initial value problem on $\mathbb{R}^d$

$$\begin{aligned} \frac{\partial\phi}{\partial t}(x, t) &= \left[(1 - \alpha)\kappa(x, t) - \alpha D(x)\right]|\nabla\phi(x, t)|\,, \\ \phi(x, 0) &= \phi_0(x)\,. \end{aligned} \tag{5}$$

In Section 3 the numerical solution of the level set PDE (5) with speed function (4) is discussed.

3 Discretisation

The first order scheme of Lefohn et al. [11] for the 2D problem is presented, with the 3D case being a straightforward generalisation. Using standard notation $\phi^n_{i,j} = \phi(i\Delta x, j\Delta y, n\Delta t)$, one has the forward Euler scheme

$$\phi^{n+1}_{i,j} = \phi^n_{i,j} - \Delta t F^n_{i,j} |\nabla \phi^n_{i,j}| \,, \tag{6}$$

where $F^n_{i,j} = \alpha D_{i,j} - (1-\alpha)\kappa^n_{i,j}$.

Assuming that $\Delta x = \Delta y = 1$, a standard first order upwind scheme [11, 14] was used to discretise the gradient operator in (5). Upwind schemes choose an appropriate spatial finite difference operator depending on the characteristics of the underlying PDE and are ubiquitous in the numerical solution of hyperbolic PDEs. Here, for the discretisation of $|\nabla\phi|$, if $F^n_{i,j} > 0$, then one uses

$$\begin{aligned} |\nabla \phi^n_{i,j}|^2 = {} & \max(\phi^n_{i+1,j} - \phi^n_{i,j}, 0)^2 + \min(\phi^n_{i,j} - \phi^n_{i-1,j}, 0)^2 \\ & + \max(\phi^n_{i,j+1} - \phi^n_{i,j}, 0)^2 + \min(\phi^n_{i,j} - \phi^n_{i,j-1}, 0)^2 \,, \end{aligned} \tag{7}$$

and if $F^n_{i,j} < 0$, then one uses

$$\begin{aligned} |\nabla \phi^n_{i,j}|^2 = {} & \min(\phi^n_{i+1,j} - \phi^n_{i,j}, 0)^2 + \max(\phi^n_{i,j} - \phi^n_{i-1,j}, 0)^2 \\ & + \min(\phi^n_{i,j+1} - \phi^n_{i,j}, 0)^2 + \max(\phi^n_{i,j} - \phi^n_{i,j-1}, 0)^2 \,. \end{aligned} \tag{8}$$

The curvature component is a parabolic term and could be approximated using central differences [14]. However, the difference of normals method is used as it is claimed that the central difference approach can lead to instabilities for more complicated speed functions which will be considered in future work [11]. This method computes normals at half grid points and takes first order differences to approximate their rate of change.

First the normals are computed:

$$n^+ = \begin{pmatrix} \dfrac{\phi^n_{i+1,j}-\phi^n_{i,j}}{\sqrt{\left(\phi^n_{i+1,j}-\phi^n_{i,j}\right)^2+\left(\frac{\phi^n_{i+1,j+1}-\phi^n_{i+1,j-1}+\phi^n_{i,j+1}-\phi^n_{i,j-1}}{4}\right)^2}} \\ \dfrac{\phi^n_{i,j+1}-\phi^n_{i,j}}{\sqrt{\left(\phi^n_{i,j+1}-\phi^n_{i,j}\right)^2+\left(\frac{\phi^n_{i+1,j+1}-\phi^n_{i-1,j+1}+\phi^n_{i+1,j}-\phi^n_{i-1,j}}{4}\right)^2}} \end{pmatrix}, \tag{9}$$

$$n^- = \begin{pmatrix} \dfrac{\phi^n_{i,j}-\phi^n_{i-1,j}}{\sqrt{\left(\phi^n_{i,j}-\phi^n_{i-1,j}\right)^2+\left(\frac{\phi^n_{i-1,j+1}-\phi^n_{i-1,j-1}+\phi^n_{i,j+1}-\phi^n_{i,j-1}}{4}\right)^2}} \\ \dfrac{\phi^n_{i,j}-\phi^n_{i,j-1}}{\sqrt{\left(\phi^n_{i,j}-\phi^n_{i,j-1}\right)^2+\left(\frac{\phi^n_{i+1,j-1}-\phi^n_{i-1,j-1}+\phi^n_{i+1,j}-\phi^n_{i-1,j}}{4}\right)^2}} \end{pmatrix}. \tag{10}$$

The approximation to the curvature is then

$$\kappa^n_{i,j} = (n^+_x - n^-_x) + (n^+_y - n^-_y)\,. \tag{11}$$

The first order forward Euler time stepping of (6) introduces stability concerns.[1] Here, adaptive time stepping is used to choose $\Delta t(t)$ such that the CFL condition is always met. For a problem on an artibrary dimensional domain, the CFL condition in terms of the normal speed is

$$\Delta t(t) < \frac{C}{F_{\max}(t)}\,, \tag{12}$$

where $F_{\max}(t) = \max_{x\in\mathcal{X}} |F_\Theta(x,t)|$ and $C = 0.9 < 1$. The condition (12) ensures that the speed computed at each pixel does not exceed one pixel per time step, and thus guarantees that the mathematical domain of dependence is contained in the numerical domain of dependence.

Finally, on a finite domain the question of boundary conditions arises. An additional ring of artificial or 'ghost' cells/pixels was added around the computational domain and assigned the values of their nearest neighbour in the original image at each time step.

[1]Semi-Lagrangian methods were introduced in the context of level set PDEs [17]. These are are explicit yet unconditionally stable and will be the subject of further investigation.

4 OpenCL implementation

In OpenCL the global problem space is broken into individual work items (threads) which execute instances of kernel functions on the GPU's compute units (CUs). Work items are collected together to form work groups (thread blocks) which are then distributed across available CUs. Within each work group, work items share fast local memory which provides a type of cache for the slower global memory (approximately 50–100 times faster memory operations). Individual work items also have access to private memory (registers) local to the CU on which they execute.

Work items within work groups are executed in a SIMD fashion in blocks of 32 threads called 'warps'. If two threads executing in the same warp have different branches of execution, then the execution of that warp is serialised and a phenomenon called 'warp divergence' occurs. This problem plagues upwind schemes as neighbouring pixels can have different upwind directions, as in equations (7) and (8). To prevent warp divergence, multiplicative masks are applied to remove conditional branches in the kernel.

The number of work groups concurrently executing per CU is limited by the availability of registers and local memory. These limits can result in under utilisation of CUs and a reduction in performance, particularly for memory bound kernels. The work group size must therefore be carefully chosen to achieve optimal performance and hide latency associated with memory transfers. An additional complication is that the work group size must evenly divide the global problem size. The original input data is padded where appropriate to avoid this problem, a warp divergence free alternative to the common technique of enqueueing the kernel with an inflated global size and masking fictitious work items. Finally, since the unit of execution is a warp, the work group size should be chosen to be a multiple of the warp size to maximise throughput.

Data transfer between CPU and GPU is between CPU RAM and GPU global memory over PCI-E, and can constitute a performance bottleneck. Iterative schemes which require only an initial data transfer (of ϕ_0) from host CPU to GPU and a final data transfer to collect the results minimise this overhead. Therefore, with a fixed time step Δt, the finite difference scheme outlined in Section 3 is well suited for GPU implementation, as one need only write a kernel which applies the finite difference stencil to an *single* pixel $\mathbf{x}$ to update $\phi(\mathbf{x}, t)$ to $\phi(\mathbf{x}, t + \Delta t(t))$. Furthermore, the purely input image dependent component of the speed term is pre-computed and loaded into GPU global memory before the iterations begin.

The adaptive scheme using (12) presents some difficulty as the global maximum pixel speed must be computed at every iteration before the update in (6) can occur. Due to the associative and commutative nature of the maximum operator, a parallel reduction algorithm [8] which can be efficiently implemented on the GPU is used, in conjunction with two additional kernels for compute and updating, providing the necessary global synchronisation.

First, the compute kernel computes the curvature (9, 10, 11), speed (6), and gradient magnitude (7, 8) at a particular pixel, writing $F^n_{i,j}\left|\nabla\phi^n_{i,j}\right|$ and $\left|F^n_{i,j}\right|$ to global memory buffers. The reduction kernel then computes $F_{\max}(t)$ in-place from the global memory buffer containing $\left|F^n_{i,j}\right|$, using local memory optimisations such as sequential addressing to avoid memory bank conflicts. Next, the host CPU reads the reduction result and computes the maximum CFL satisfying time step $\Delta t(t)$ according to equation (12), before enqueueing the final kernel to perform the update from t to $t+1$ in (6) by re-using the buffer consisting of the product $F^n_{i,j}\left|\nabla\phi^n_{i,j}\right|$. Finally, the required boundary extension is simply performed by enqueueing the relevant kernels with work item offsets and performing copy instructions directly between the GPU buffers.

5 Results

Figure 1 illustrates the effect of changing α on the segmentation results for a 2D cross section of a femural bone taken from a micro-CT scan of a mouse [4].

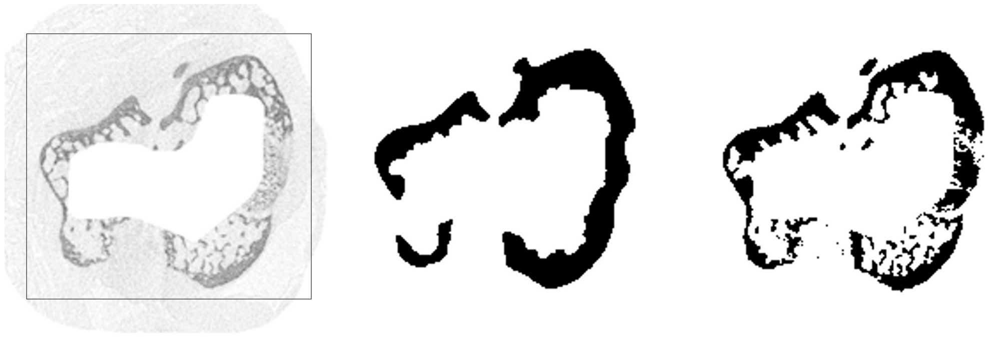

Figure 1: The effect of α on the final segmentation with $T = 85$, $\epsilon = 45$. (Left) Image from micro-CT scan, with the initial front used for computing ϕ_0. (Middle) Curvature dominated, $\alpha = 0.01$. (Right) Advection dominated, $\alpha = 0.99$. Data courtesy of Prof. Peter Croucher of the Garvan Institute.

The mouse was suffering from multiple myeloma, a rare type of bone marrow cancer which results in holes forming in the bone. The results were said to have converged when the number of pixels changing in the final mask between time steps was at most two for at least ten consecutive iterations, though many other criteria have been suggested [3].

Figures 2 and 3 illustrate the effect of work group size on the average speed-up achieved by the OpenCL implementation in the 2D case. The average speed-up of the adaptive and non-adaptive time stepping methods are compared for different hardware over 1000 iterations. The reinitialisation procedure is not included in the profiling results; only the core computation is profiled. It is also assumed that the GPU buffers were appropriately initialised. Furthermore, the work group size for the reduction kernel was fixed at 128 work items, as this was found to be optimal for both hardware configurations. The work group size corresponds to the compute/update kernels in the adaptive case.

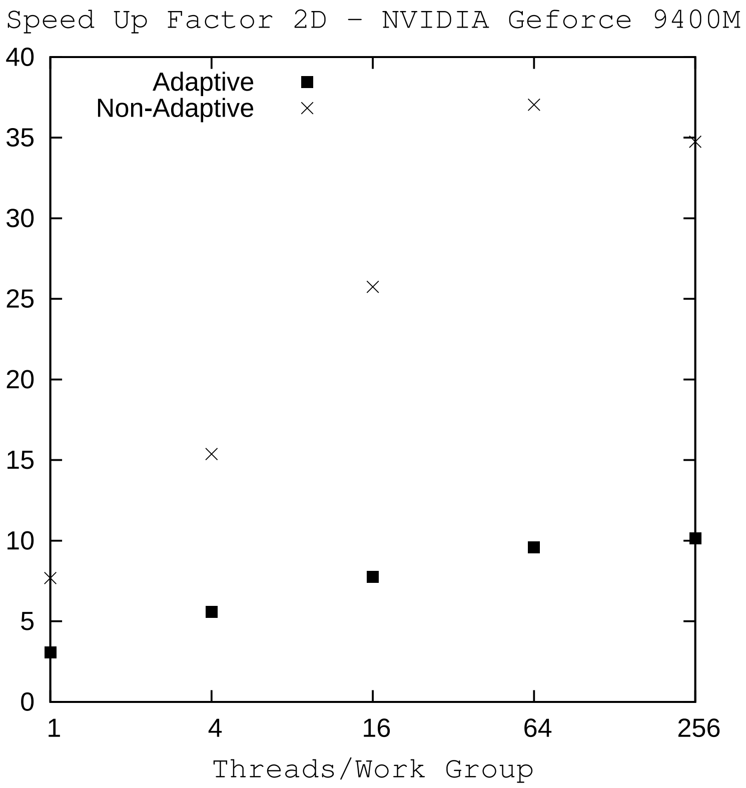

Figure 2: Host CPU: 2.53 GHz Intel Core 2 Duo, 4 GB 1067 MHz DDR3, GPU: NVIDIA Geforce 9400M, $\bar{\text{CPU}} = 9.37$ s (non-adaptive) and 9.56 s (adaptive).

The average speed-up is

$$\bar{S} := \frac{\bar{\text{CPU}}}{\bar{\text{GPU}}} = \frac{\frac{1}{N}\sum_{i=1}^{N} \text{CPU}_i}{\frac{1}{N}\sum_{i=1}^{N} \text{GPU}_i}, \tag{13}$$

where $\text{CPU}_1, \ldots, \text{CPU}_N$ are independent samples of the CPU execution time, $\text{GPU}_1, \ldots, \text{GPU}_N$ are independent samples of the GPU execution time, and thus $\bar{\text{CPU}}$ and $\bar{\text{GPU}}$ are independent.

Both the adaptive and non-adaptive algorithms provide significant speed-ups

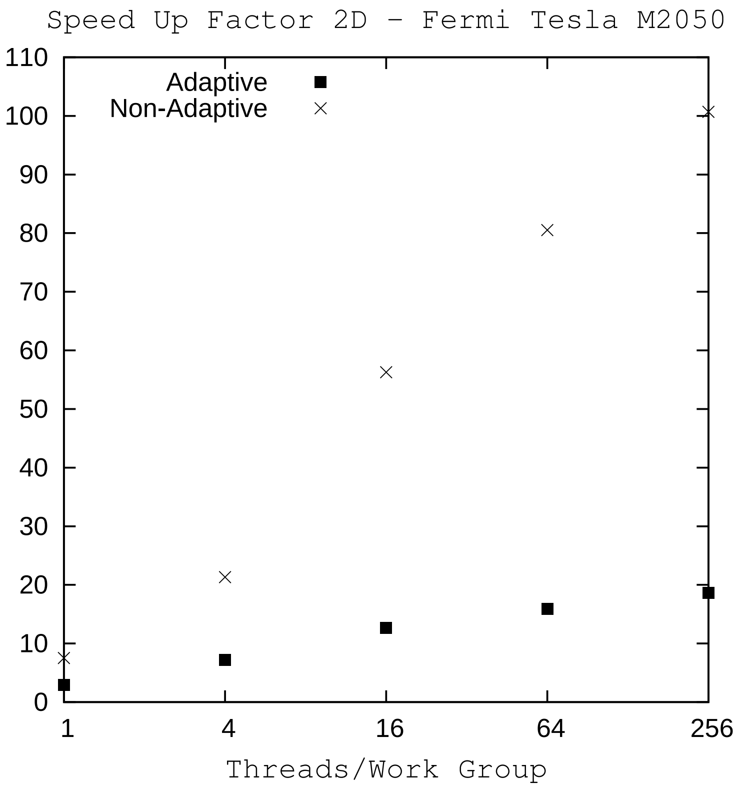

Figure 3: Host CPU: Dual Xeon 8-core E5-2650, GPU: NVIDIA Fermi Tesla M2050, $\bar{\text{CPU}} = 7.65\,\text{s}$ (non-adaptive) and $7.80\,\text{s}$ (adaptive).

when implemented on the GPUs using optimisations described in Section 4 and an appropriate work group size. In particular, note the improved performance when the work group size is a multiple of the warp size, in this case 32.

Figures 4 and 5 show the acceleration factors achieved for both algorithms when operating on a subset of the full 3D mouse data set (of dimension $384 \times 387 \times 41$) over 25 iterations. The full data set ($384 \times 397 \times 346$) was not used as it exceeded the maximum global memory buffer size of the Geforce 9400M. Although multiple kernel invocations combined with host GPU data transfers

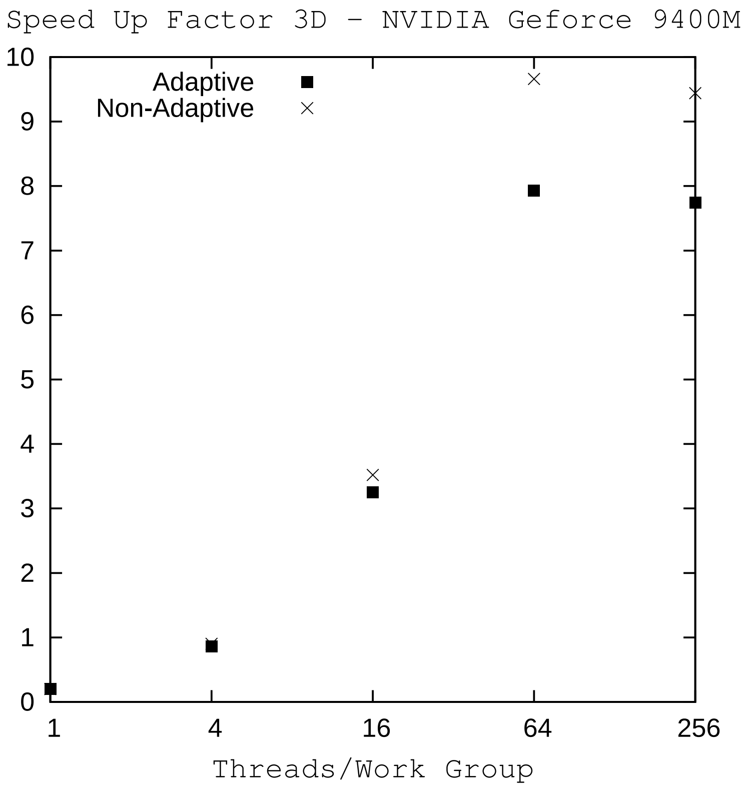

Figure 4: Host CPU: 2.53 GHz Intel Core 2 Duo, 4 GB 1067 MHz DDR3, GPU: NVIDIA Geforce 9400M. $\bar{\text{CPU}} = 42.95$ s (non-adaptive) and 45.07 s (adaptive).

can be used to address this issue [9], a more efficient approach is to invoke multiple GPUs in a cluster environment using Open MPI. A comparison of these two approaches is the subject of future work.

Further speed-ups might be expected with local memory usage during the core compute, as currently only the reduction kernel employs local memory. Nonetheless it has been demonstrated that interactive speeds are achievable with OpenCL.

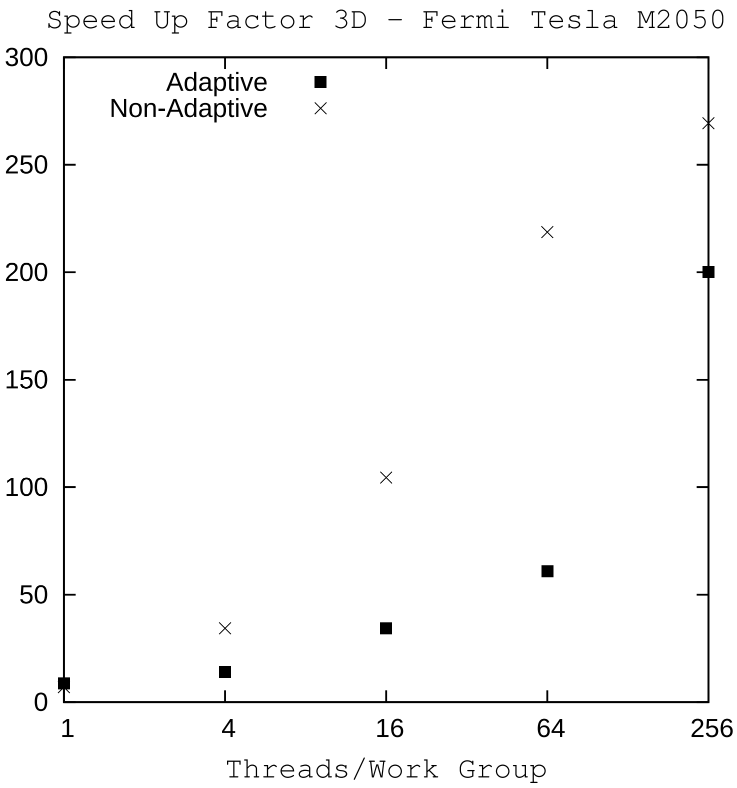

Figure 5: Host CPU: Dual Xeon 8-core E5-2650, GPU: NVIDIA Fermi Tesla M2050, $\bar{\text{CPU}} = 34.98$ s (non-adaptive) and 37.40 s (adaptive).

References

[1] D. Adalsteinsson and J. A. Sethian. The fast construction of extension velocities in level set methods *J. Comp. Phys.* 148(1):2–22, 1999. doi:10.1006/jcph.1998.6090 C331

[2] J. F. Aujol and G. Aubert. Signed distance functions and viscosity solutions of discontinuous Hamilton–Jacobi equations *Research Report, Inria, France* 4507, 2002. C331

[3] K. N. Chaudhury and K. R.Ramakrishnan. Stability and convergence of the level set method in computer vision *Patt. Rec. Lett.* 28:884–893, 2007. doi:10.1016/j.patrec.2006.12.003 C337

[4] S. De Weerdt. Animal models: Towards a myeloma mouse *Nature* 480(7377):S38–S39, 2011. doi:10.1038/480S38a C336

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions *Cornell Comp. Inf. Sci.* TR2004-1963, 2004. C331

[6] J. Gomes and O. Faugeras. Reconciling distance functions and level sets *J. Vis. Comm. Image Rep.* 11:209–223, 2000. doi:10.1006/jvci.1999.0439 C330, C331

[7] Khronos Group. The open standard for parallel programming of heterogeneous systems. `http://www.khronos.org/opencl/`, 17/10/2012. C329

[8] M. Harris et al. Optimizing parallel reduction in CUDA *Proc. of ACM SIGMOD* 21(13):104–110, 2007. C336

[9] D. Lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky and A. W. Toga. CUDA optimization strategies for compute—and memory—bound neuroimaging algorithms *Comp. Meth. Prog. Biomed.* 106:175–187, 2012. doi:10.1016/j.cmpb.2010.10.013 C329, C340

[10] C. Li, C. Xu, C. Gui and M. D. Fox. Level set evolution without re-initialization: a new variational formulation *IEEE Conf. Comp Vision and Pattern Recognition* 1:430–436, 2005. doi:10.1109/CVPR.2005.213 C330, C331

[11] A. E. Lefohn, J. M. Kniss, C. D. Hansen and R. T. Whitaker. A streaming narrow-band algorithm: interactive computation and visualization of level sets *IEEE Trans. Vis Comp Graphics* 10(4):422–433, 2004. doi:10.1109/TVCG.2004.2 C329, C330, C332, C333

[12] R. Malladi, J. A. Sethian and B. C. Vermuri. Shape modeling with front propagation: a level set approach *IEEE Trans. Patt. Analy. Mach. Int.* 17(2):158–175, 1995. doi:10.1109/34.368173 C330, C332

[13] S. Osher and N. Paragios. *Geometric level set methods in imaging, vision, and graphics*. Springer 2003. C330, C331

[14] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations *J. Comp. Phys.* 79:12–49, 1988. doi:10.1016/0021-9991(88)90002-2 C330, C331, C333

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn and T. J. Purcell. A survey of general-purpose computation on graphics hardware *Computer Graphics Forum* 26(1):80–113, 2007. doi:10.1111/j.1467-8659.2007.01012.x C329

[16] J. A. Sethian. *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematics (3), 2nd Ed, 1999. C330, C331

[17] J. Strain. Semi-Lagrangian methods for level set equations *J. Comp. Phys* 151(2):498–533, 1999. doi:10.1006/jcph.1999.6194 C334

[18] K. R. Varshney and A. S. Willsky. Classification using geometric level sets *J. Machine Learning Research* 11:491–516, 2010. `http://dl.acm.org/citation.cfm?id=1756006.1756020` C330

Author addresses

1. **M. J. Piggott**, CSIRO Mathematics, Informatics and Statistics, North Ryde NSW 2113, Australia
`mailto:marc.piggott@csiro.au`

2. **P. Vallotton**, CSIRO Mathematics, Informatics and Statistics, North Ryde NSW 2113, Australia
mailto:pascal.vallotton@csiro.au

3. **J. A. Taylor**, CSIRO Mathematics, Informatics and Statistics, Acton ACT 2601, Australia
mailto:john.a.taylor@csiro.au

4. **T. P. Bednarz**, CSIRO Mathematics, Informatics and Statistics, North Ryde NSW 2113, Australia
mailto:tomasz.bednarz@csiro.au