Geometric Multigrid on Multicore Architectures: Performance-Optimized Complex Diffusion

M. Stürmer, H. Köstler, and U. Rüde

Lehrstuhl für Systemsimulation
Friedrich-Alexander-Universität Erlangen-Nürnberg
EMG 2008

October 21, 2008
Image Denoising
by Diffusion

Figure: Denoising of a 2D CT slice (data: Siemens AG, Healthcare Sector, CT division).
Contents

Application

Cache Blocking

Results

Future Work
Noise Model

Assumption:
Relation between an original, unknown image $u : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ and an observed image $u^0$ can be expressed by

$$u^0 = u + \eta,$$

where $\eta$ stands for the noise.

We assume white additive Gaussian noise.
Denoising by Diffusion

Idea:
Use nonlinear anisotropic diffusion process to denoise the image $u^0$, i.e. solve the time-dependent PDE

\[
\begin{align*}
\text{div}(g \nabla u) &= \frac{\partial u}{\partial t} \quad \text{in } \Omega \times \mathbb{T} \\
\langle g \nabla u, n \rangle &= 0 \quad \text{on } \partial\Omega \times \mathbb{T} \\
u(x, 0) &= u^0(x) \quad \text{in } \Omega.
\end{align*}
\]
Complex Diffusion

- extend the nonlinear diffusion model by using a complex diffusivity function $g(u) : \mathbb{C} \mapsto \mathbb{C}$ with
  \[ g(u) = \frac{e^{i\theta}}{1 + \left(\frac{Im(u)}{k\theta}\right)^2} \]
- the parameter $\theta$ denotes a small angle, $k$ is a soft threshold
- the solution of the anisotropic diffusion PDE becomes complex
- $u(x, t)$’s real part is the denoised image, its imaginary part acts as an edge detector
- improved results for ramp type edges [GSZ04]
Multigrid for Complex Diffusion

time-dependent, complex, nonlinear diffusion PDE:
  • spatial discretization by finite volumes
  • semi-implicit time discretization
  • the nonlinear diffusion is handled by inexact lagged diffusivity
  • cell-based FAS multigrid
    • damped Jacobi smoother
    • standard transfers
Image Denoising Result

(a) Noisy image  (b) Denoised image  (c) Imaginary part

Figure: Denoising of a test image with added Gaussian noise.
Results and Performance

Run-time for denoising a $4096 \times 4096$ image

![Chart showing run-time for denoising a 4096x4096 image with different numbers of CPUs. The chart shows a significant reduction in run-time as the number of CPUs increases.]
Results and Performance

Run-time for denoising a $4096 \times 4096$ image

Real-time performance required!
The DiME Project\(^1\) (Data-local Iterative MEthods)

3D Poisson Red-Black GS smoother on Itanium 2 [STR08]

\(^1\)http://www10.informatik.uni-erlangen.de/Research/Projects/DiME/
Cache Blocking Techniques

Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Without Cache Blocking:
Cache Blocking Techniques

Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Without Cache Blocking:
Cache Blocking Techniques
Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Without Cache Blocking:
Cache Blocking Techniques
Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Simple Line Blocking:
Cache Blocking Techniques

Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Simple Line Blocking:
Cache Blocking Techniques

Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Simple Line Blocking:
Cache Blocking Techniques

Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Simple Line Blocking:
Cache Blocking Techniques

Red-Black Gauss-Seidel Smoother with 5-Point-Stencil

Simple Line Blocking:
What has changed since then

Two workstations:

<table>
<thead>
<tr>
<th>Year</th>
<th>2004</th>
<th>2008</th>
<th>factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU(s)</td>
<td>1× P4 Prescott</td>
<td>2× C2 Penryn</td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>3.2 GHz</td>
<td>2.8 GHz</td>
<td>0.88</td>
</tr>
<tr>
<td>Stream Triad²</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>serial</td>
<td>2.9 GB/s</td>
<td>3.6 GB/s</td>
<td>1.24</td>
</tr>
<tr>
<td>OMP</td>
<td>(2.6 GB/s)</td>
<td>7.6 GB/s</td>
<td>2.62</td>
</tr>
<tr>
<td>FP performance³</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scalar</td>
<td>4.8 GFlop/s</td>
<td>8×5.6 GFlop/s</td>
<td>1.17 (9.33)</td>
</tr>
<tr>
<td>SIMD</td>
<td>12.8 GFlop/s</td>
<td>8×22.4 GFlop/s</td>
<td>1.75 (14)</td>
</tr>
</tbody>
</table>

² gcc 4.3.2 -O3 -m32 -march=prescott
³ single precision
What has changed since then

**Parallelization**
- serial programs *are slow*
- parallelization has become mandatory

**Memory Efficiency**
- available bandwidth per core decreases
- memory-efficiency increasingly important
- temporal cache blocking promising

**Optimized Kernels**
- huge computational power, especially in SIMD
- kernels need not be optimized that aggressively
- SIMD vectorization is a key optimization
Cache Blocking Techniques

First sweep on finest grid downwards the V(1,1)

How we did it:

source
target
buffer

A B C D E
A B C D E
A B C D E
Cache Blocking Techniques
First sweep on finest grid downwards the V(1,1)

How we did it:

Copy to buffer:
- approximation
- RHS
Cache Blocking Techniques
First sweep on finest grid downwards the V(1,1)

How we did it:

source

Compute:
- calc. coeff.
- red update
- calc. coeff.
- black update
- calc. coeff.
- residual
- restrict res.
- restrict sol.

buffer

target

A B C D E

A B C D E

A B C D E
Cache Blocking Techniques
First sweep on finest grid downwards the V(1,1)

How we did it:
Cache Blocking Techniques
First sweep on finest grid downwards the V(1,1)

How we did it:

source

buffer

A B

Compute:
- calc. coeff.
- red update
- calc. coeff.
- black update
- calc. coeff.
- residual
- restrict res.
- restrict sol.

A B C D E

A B C D E

target
Cache Blocking Techniques
First sweep on finest grid downwards the V(1,1)

How we did it:

Copy to buffer:
- approximation
- RHS

Copy to main memory:
- approximation
- coarse residual
- coarse solution

source

buffer

target
Cache Blocking Techniques
First sweep on finest grid downwards the V(1,1)

How we did it:

source

buffer

Compute:
- calc. coeff.
- red update
- calc. coeff.
- black update
- calc. coeff.
- residual
- restrict res.
- restrict sol.

target
Parallel Cache Blocking with Buffers

Pros and Cons

- data dependencies between neighboring tiles
- access only to a small window of data
+ simple parallelization
+ low synchronization overhead
+ temporary data (like coefficients) can be handled efficiently
  - separate data movement and computation
    + data layout optimizations, fast addressing in buffers
    + optimized load / store instructions
    + separate move / compute threads operating on shared caches
    + direct applicable on the Cell processor
- copying needs extra time
+ can be well put into a framework
Results and Performance

Performance of optimized multigrid

Run-time for denoising a $4096 \times 4096$ image

![Bar chart showing run-time for denoising a $4096 \times 4096$ image with different numbers of CPUs. The chart compares reference, blocking, and optimized run-times.]
## Results and Performance

Performance of optimized multigrid

<table>
<thead>
<tr>
<th></th>
<th>run-time [s]</th>
<th>GFlop rate eff.</th>
<th>internal</th>
<th>bandwidth [GB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocking</td>
<td>2.47</td>
<td>15.4</td>
<td>16.3</td>
<td>3.0</td>
</tr>
<tr>
<td>optimized</td>
<td>1.52</td>
<td>25.1</td>
<td>26.6</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Computational density:

- 326.5 Flop per unknown on finest grid
- 383.5 Flop per unknown on other grids
Image Denoising

Figure: Denoising of a CT slice.

- using 5 V(2,2)-cycles with $t = 1$, $\theta = 1^\circ$, and $k = 0.005$
- reduction of residual $2 \cdot 10^6$
Future Work

- **Modelling**
  - compare against patch- and wavelet-based denoising methods
  - include a priori knowledge, estimate noise distribution
- **Framework**
  - increase usability and clean up code
  - portage to the Cell Broadband Engine Architecture
  - extend to 3D
  - compare against GPU implementation
References

G. Gilboa, N. Sochen, and Y.Y. Zeevi.
Image Enhancement and Denoising by Complex Diffusion Processes.

Optimizing a 3D Multigrid Algorithm for the IA-64 Architecture.
*International Journal of Computational Science and Engineering (IJCSE)*, accepted for publication 2008.
Thank you very much for your attention!