Cache Optimizations for Multigrid Codes

Markus Kowarschik, Ulrich Rüde, Nils Thürey
kowarschik@cs.fau.de, ruede@cs.fau.de, thuerey@cs.fau.de
Lehrstuhl für Systemsimulation (Informatik 10)
Institut für Informatik
Friedrich–Alexander–Universität Erlangen–Nürnberg
Germany

Christian Weiß
weissc@cs.tum.edu
Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR–TUM)
Fakultät für Informatik
Technische Universität München
Germany
Outline

• Motivating example

• Cache design issues

• Cache optimizations for multigrid codes
  – Data layout optimizations
  – Data access optimizations

• Conclusions
Motivating (frustrating?) example

*Theoretically* ... modern workstations based on superscalar RISC processors can do by far more than 1000 MFLOPS.

*In practice* ... we often obtain disappointing results.

*Example:* 3D Gauss–Seidel iteration on a Digital PWS 500au, constant coefficients

<table>
<thead>
<tr>
<th>grid size</th>
<th># unknowns</th>
<th>MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4096</td>
<td>415</td>
</tr>
<tr>
<td>32</td>
<td>32768</td>
<td>194</td>
</tr>
<tr>
<td>64</td>
<td>262144</td>
<td>76</td>
</tr>
<tr>
<td>128</td>
<td>$≈ 2.1 \cdot 10^6$</td>
<td>73</td>
</tr>
</tbody>
</table>

⇒ Need to understand cache effects!
Cache design issues — a memory hierarchy example

Digital PWS 500au memory architecture:

```
<table>
<thead>
<tr>
<th>Capacity</th>
<th>Bandwidth</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 Bytes</td>
<td>24000 MB/s</td>
<td>2 ns</td>
</tr>
<tr>
<td>8 KBytes</td>
<td>16000 MB/s</td>
<td>2 ns</td>
</tr>
<tr>
<td>96 KBytes</td>
<td>8000 MB/s</td>
<td>6 ns</td>
</tr>
<tr>
<td>2 MBytes</td>
<td>888 MB/s</td>
<td>24 ns</td>
</tr>
<tr>
<td>1536 MBytes</td>
<td>1000 MB/s</td>
<td>112 ns</td>
</tr>
</tbody>
</table>
```

Swap Space on Disk

Exploit the cache architecture more efficiently!
Cache design issues — associativity

Denotes the number of cache lines where a main memory block may be copied to

Low associativity (n small) $\Rightarrow$ High potential for cache conflict misses, cache thrashing
Techniques to enhance cache utilization

- **Data layout optimizations:**
  Address data storage schemes in memory

- **Data access optimizations:**
  Address the order in which the data are accessed
Data layout optimizations — cache–aware data structures

This is the particular focus of our paper, more details are provided therein

_Idea:_ Merge data which are needed together to increase _spatial locality:_ cache lines contain several data items

_Example:_ Gauss–Seidel on $Au = f$, 2D, 5–point stencils:

$$u^{(k+1)}_i = a_{i,i}^{-1} \left( f_i - \sum_{j<i} a_{i,j} u^{(k+1)}_j - \sum_{j>i} a_{i,j} u^{(k)}_j \right), \quad i = 1, \ldots, N$$

typedef struct {
    double f;
    double cCenter, cNorth, cEast, cSouth, cWest;
} eqnData;

double u[N][N]; // Solution vector
eqnData rhsAndCoeff[N][N]; // Right–hand side and coefficients
Data layout optimizations — array padding

*Idea:* Increase array dimensions to change relative distances between elements ⇒ Avoid severe cache conflict misses; e.g., in stencil computations

*Example:* 2D array, FORTRAN77 (column major ordering)

double precision $u(1024, 1024)$ becomes
double precision $u(1024+\text{pad}, 1024)$, *problem:* pad = ?
Data layout optimizations — array padding

In 3D we use a non-standard array padding approach

Example: FORTRAN77 implementation

double precision u(0:n+pad1,0:n,0:n), dummy(0:n*pad2-1)
do k = 1, n-1
  do j = 1, n-1
    do i = (k*pad2) + 1, (k*pad2) + n-1
      RELAX(i,j,k) // pad2 needs to be respected here as well
    enddo
  enddo
enddo
Padding approaches:

- **Analytic/Algebraic techniques (Rivera/Tseng)**
  - Block size (tile size) and paddings depend on array size and cache capacity
  - Often not general enough for realistic problems where several arrays are involved; e.g., CFD: pressure, velocity field, temperature, concentrations of chemical species, etc.

- **Exhaustive parameter search**
  - **AEOS paradigm**: *Automated Empirical Optimization of Software*
  - Examples:
    * **ATLAS (Automatically Tuned Linear Algebra Software)**
    * **FFTW (The Fastest Fourier Transform in the West)**
  - Searching the parameter space is time-consuming, but currently the most promising cache tuning approach!
**Data access optimizations — loop blocking (loop tiling)**

*Idea:* Divide the iteration space into blocks and perform as much work as possible on the data in cache (i.e., on the current *block*) before switching to the next block
⇒ Enhance spatial and/or temporal locality

*Popular textbook example:* Matrix multiplication

**Before loop blocking:**

```plaintext
do J= 1,N
do K= 1,N
do I= 1,N
   \( C(I,J) = C(I,J) + A(I,K) \times B(K,J) \)
enddo
enddo
enddo
```

**After loop blocking:**

```plaintext
do KK= 1,N,W // W = tile width
do II= 1,N,H // H = tile height
do J= 1,N
   do K= KK,min(KK+W-1,N)
      do I= II,min(II+H-1,N)
         \( C(I,J) = C(I,J) + A(I,K) \times B(K,J) \)
      enddo
   enddo
enddo
enddo
```
Data access optimizations — loop blocking

Blocking is also possible for iterative methods for linear systems.

Blocking the iteration loop means merging successive iterations into a single pass through the data set ⇒ Enhance cache reuse.

*Example:* Red/black Gauss–Seidel smoother

Data dependencies need to be respected.

Similar techniques for all stencil–based methods.
There are even more alternatives in 3D; e.g., red/black Gauss-Seidel smoother

There are even more alternatives in 3D; e.g., red/black Gauss-Seidel smoother.

- **a)** Fused Loops
- **b)** 1-Way Blocked Iterations

- **i**
  - Relax red points
  - Relax black points
  - Relax red points for the second time
  - Relax black points for the second time

- **i-1**
  - Relax red points
  - Relax black points

- **i-2**
  - Relax red points for the second time
  - Relax black points

- **i-3**
  - Relax red points for the second time
  - Relax black points

- **2**
  - Red point relaxed twice
  - Black point relaxed twice

- **black point relaxed once**
- **red point relaxed once**

- **2**
  - Points to relax
Data access optimizations — other techniques

There is a variety of other data access optimizations

- *Loop interchange*: lessen the impact of non–unit stride accesses
- *Loop fusion*: reduce the number of sweeps through the data set ⇒ Increase temporal locality
- *Data copying*: copy non–contiguous data to contiguous memory locations ⇒ Reduce cache conflicts and/or drops in performance due to limited TLB capacity
- etc.
Speedups for 2D Gauss–Seidel smoother, constant coefficients (left side: Alpha 21164, right side: Alpha 21264, HP PA 8500)

Variable–coefficient problems: speedup factors of 2–3 can be obtained for large grids, the MFLOPS rates are usually smaller since much more data have to be loaded.

Current research efforts focus on the 3D case, where TLB effects become more dramatic than in 2D.
Conclusions and final remarks

We have investigated cache performance optimizations for structured grid multigrid

- in 2D and 3D
- for the constant coefficient and for the variable coefficient case

DiME project: data–local iterative methods for the efficient solution of PDEs

There’s a cache–optimized multigrid library which can be downloaded from the web site given below: DiMEPACK

We are currently investigating how these techniques can be integrated into more involved methods; e.g., adaptive multigrid

More details are available:

http://www10.informatik.uni-erlangen.de/dime