CS 294-73
Software Engineering for Scientific Computing

Lecture 18: Performance Measurements for Multigrid
Multigrid

\textit{vcycle}(\phi, \rho)
\{
    \phi := \phi + \lambda(L(\phi) - \rho) \ p \times \\
    \text{if (level > 0)}
    \{
        \mathcal{R} = \rho - L(\phi) \\
        \mathcal{R}_c = \mathcal{A}(\mathcal{R}) \\
        \delta : B_c \rightarrow \mathbb{R}, \delta = 0 \\
        \text{vcycle}(\delta, \mathcal{R}_c) \\
        \phi := \phi + \mathcal{I}(\delta) \\
        \phi := \phi + \lambda \ast (L(\phi) - \rho) \ p \times \\
    \}
\} \\
\text{else}
\{
}\{
    \phi := \phi + \lambda \ast (L(\phi) - \rho) \ p_B \times \\
\}
\}

At the top level, iterate until residual is reduced by some large factor.
Case Study

• 2D, 1024x1024 grid, 10 iterations.

• Focus on different versions of computing the residual. 8 flops per grid point.

• -O3, SIMD reporting turned on.
Multigrid v-cycle.

Multigrid::vCycle(...)
{
  ...
  if (m_level > 0)
  {
    pointRelax(a_phi,a_rhs,m_preRelax);
    residual(m_res,a_phi,a_rhs);
    avgDown(m_resc,m_res);
    m_delta.setVal(0.);
    m_coarsePtr->vCycle(m_delta,m_resc);
    fineInterp(a_phi,m_delta);
    pointRelax(a_phi,a_rhs,m_postRelax);
  }
  else
  pointRelax(a_phi,a_rhs,m_bottomRelax);
}
What are timers reporting?

- A separate timer for every call in a call stack. For the recursive calls in multigrid, this gives a disaggregated picture of performance.

[2] MG top level 5.87228 10
  41.9% 2.4576 10 vcycle [3]
  ...
  90.3% Total

-------------------------------------------------------------

[3] vcycle 2.45764 10
  56.5% 1.3888 10 residual [7]
  24.4% 0.6001 10 vcycle [8]
  17.7% 0.4361 20 relax [9]
  0.7% 0.0182 10 fineInterp [23]
  0.5% 0.0120 10 avgdown [29]
  0.1% 0.0025 10 BoxData::setval [59]
  100.0% Total

-------------------------------------------------------------
What are timers reporting?

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcycle</td>
<td>0.60009</td>
<td>10 residual [11]</td>
</tr>
<tr>
<td></td>
<td>57.6%</td>
<td>0.3457</td>
</tr>
<tr>
<td></td>
<td>26.6%</td>
<td>0.1595</td>
</tr>
<tr>
<td></td>
<td>14.6%</td>
<td>0.0876</td>
</tr>
<tr>
<td></td>
<td>0.6%</td>
<td>0.0037</td>
</tr>
<tr>
<td></td>
<td>0.5%</td>
<td>0.0029</td>
</tr>
<tr>
<td></td>
<td>0.1%</td>
<td>0.0006</td>
</tr>
<tr>
<td></td>
<td>100.0%</td>
<td>Total</td>
</tr>
</tbody>
</table>
Baseline implementation of Residual

Multigrid::residual(...)
{
    ...
    res.setVal(0.);
    for (auto it = bx.begin(); !it.done(); ++it)
        { Point pt = *pt;
          for (int dir = 0; dir < DIM ; dir++)
              { res(pt) += (a_phi(pt + e[dir]) + a_phi(pt - e[dir]));
              }
          res(pt) -= -2*DIM*a_phi(pt)
          res(pt) = res(pt)*hsqi - a_rhs(pt);
        }
}
### Time Table for Baseline

#### [3] vcycle 2.45764 10

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Time</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>56.5%</td>
<td>1.3888</td>
<td>10 residual [7]</td>
</tr>
<tr>
<td>24.4%</td>
<td>0.6001</td>
<td>10 vcycle [8]</td>
</tr>
<tr>
<td>17.7%</td>
<td>0.4361</td>
<td>20 relax [9]</td>
</tr>
<tr>
<td>0.7%</td>
<td>0.0182</td>
<td>10 fineInterp [23]</td>
</tr>
<tr>
<td>0.5%</td>
<td>0.0120</td>
<td>10 avgdown [29]</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.0025</td>
<td>10 BoxData::setval [59]</td>
</tr>
</tbody>
</table>

100.0%                  Total

---

#### [4] residual 1.42319 10

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Time</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7%</td>
<td>0.0102</td>
<td>10 BoxData::setval [34]</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.0015</td>
<td>10 getGhost [67]</td>
</tr>
<tr>
<td>0.8%</td>
<td></td>
<td>Total</td>
</tr>
</tbody>
</table>

8x1024x1024x10 = 83886080 Flops.
83886080/1.42 = 59 Mflops/sec.
Multigrid::residual(...) 
{
    ...
    double* phiptr[2*DIM+1];
    double coefs[2*DIM+1];
    a_res.setVal(0.);
    for (int q = 0; q < 2*DIM; q++)
    {
        coefs[q] = 1.0;
    }
    coefs[2*DIM] = -2.0*DIM;
for (auto it=base.begin();!it.done();++it) {
    Point pt=*it;
    for (int dir = 0; dir < DIM; dir++) {
        Point edir = Point::Basis(dir);
        phiptr[2*dir] = &a_phi(pt+mdir);
        phiptr[2*dir+1] = &a_phi(pt-mdir);
    }
    phiptr[2*DIM] = &a_phi(pt);
    double* rhsptr = &a_rhs(pt);
    double* resptr = &a_res(pt);
    for (int q = 0; q < 2*DIM+1 ; q++) {
        for (int ll=0;ll < m_domainSize; ll++) {
            resptr[ll] += phiptr[q][ll]*coefs[q];
        }
    }
    for (int ll = 0; ll < m_domainSize; ll++) {
        resptr[ll] = resptr[ll]*hsqiminus + rhsptr[ll];
    }
}
## Time Table for Pencil

<table>
<thead>
<tr>
<th>Time</th>
<th>Function</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3]v-cycle</td>
<td>0.66266</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>67.4%</td>
<td>0.4467</td>
<td>20</td>
<td>relax [4]</td>
</tr>
<tr>
<td>22.8%</td>
<td>0.1513</td>
<td>10</td>
<td>v-cycle [6]</td>
</tr>
<tr>
<td>4.9%</td>
<td>0.0327</td>
<td>10</td>
<td>residual [12]</td>
</tr>
<tr>
<td>2.7%</td>
<td>0.0177</td>
<td>10</td>
<td>fineInterp [17]</td>
</tr>
<tr>
<td>1.8%</td>
<td>0.0120</td>
<td>10</td>
<td>avgdown [25]</td>
</tr>
<tr>
<td>0.3%</td>
<td>0.0023</td>
<td>10</td>
<td>BoxData::setval [53]</td>
</tr>
<tr>
<td>100.0%</td>
<td></td>
<td></td>
<td>Total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Function</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>[12]residual</td>
<td>0.03271</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>15.9%</td>
<td>0.0052</td>
<td>10</td>
<td>BoxData::setval [38]</td>
</tr>
<tr>
<td>5.0%</td>
<td>0.0016</td>
<td>10</td>
<td>getGhost [61]</td>
</tr>
<tr>
<td>20.9%</td>
<td></td>
<td></td>
<td>Total</td>
</tr>
</tbody>
</table>

83886080 / 0.03271 = 2.56 Gflops/sec.
1.42 / 0.0327 = 43x speedup.
Proto Stencil Implementation

Multigrid::residual(
    BoxData<double >& a_res,
    BoxData<double >& a_phi,
    BoxData<double >& a_rhs
)
{
    getGhost(a_phi);
    double hsqiminus = -1.0/(m_dx*m_dx);
    a_res |= m_laplacian(a_phi,hsqiminus);
    a_res += a_rhs;
}
The stencil \( m\_\text{laplacian} \) is defined in the constructor.

\[
m\_\text{laplacian} = (-2.0*\text{DIM})*\text{Shift(getZeros())};
\]

for (int dir = 0; dir < \text{DIM} ; dir++)
{
    \text{Point edir} = \text{Point::Basis(dir)};
    \text{Stencil\<double\>} plus = 1.0*\text{Shift(ederir)};
    \text{Stencil\<double\>} minus = 1.0*\text{Shift(ederir*(-1))};

    m\_\text{laplacian} = m\_\text{laplacian} + minus + plus;
}

The apply operation for a stencil does just what we did by hand here: loop over points in the stencil, then increment the rhs by the value multiplied by the weight.
## Time Table for Stencil

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcycle</td>
<td>0.69304</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>relax</td>
<td>0.4457</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>vcycle</td>
<td>0.1594</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>residual</td>
<td>0.0558</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>fineInterp</td>
<td>0.0178</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>avgdown</td>
<td>0.0119</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>BoxData::setval</td>
<td>0.0024</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Total: 14%

---

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>residual</td>
<td>0.0558</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>BoxData::operator+=</td>
<td>0.0286</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Stencil::apply</td>
<td>0.0256</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>getGhost</td>
<td>0.0016</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Total: 100.0%

---

83886080/.0558 = 1.5 Gflops/sec.
1.42 / .0558 = 25x speedup.
.0558 / .0327 = 1.7x more time than hand-coded one.
Finer Tuning of Pencil implementation

```
for (int ll = 0; ll < m_domainSize; ll++)
{
    resptr[ll] =
        (phiptr[0][ll] + phiptr[1][ll] + phiptr[2][ll] + phiptr[3][ll]);
}

for (int ll = 0; ll < m_domainSize; ll++)
{
    resptr[ll] = (resptr[ll] - 2*DIM*phiptr[2*DIM][ll]) * hsqiminus
        + rhsptr[ll];
}
```
Finer Tuning of Pencil implementation

for (auto it=base.begin(); !it.done(); ++it)
{
...
for (int ll = 0; ll < m_domainSize; ll++)
{
    resptr[ll]=
        (phiptr[0][ll]+phiptr[1][ll]+phiptr[2][ll]+phiptr[3][ll]);
}
for (int ll = 0; ll < m_domainSize; ll++)
{
    resptr[ll] =
        (resptr[ll]-2*DIM*phiptr[2*DIM][ll])*hsqiminus + rhsptr[ll];
}
}(Note: need additional ifdef to get 3D as well).
void Multigrid::pointRelax(
    BoxData<double> & a_phi,
    BoxData<double> & a_rhs,
    int a_numIter)
{
    residual(m_res,a_phi,a_rhs);
    m_res* = -m_lambda;
    a_phi += m_res;
}
void Multigrid::pointRelax(
    BoxData<double> & a_phi,
    BoxData<double> & a_rhs,
    int a_numIter
) {
    residual(m_res,a_phi,a_rhs);
    m_res*= -m_lambda;
    a_phi += m_res;
}
Finer Tuning of Pencil implementation

[3] vcycle 0.48985 10
  67.4%  0.3299  20 relax [4]
  21.4%  0.1047  10 vcycle [6]
  4.7%   0.0230  10 residual [16]
  3.6%   0.0177  10 fineInterp [17]
  2.5%   0.0121  10 avgdown [27]
  0.5%   0.0024  10 BoxData::setval [57]
100.0%   Total

------------------------------------------

1146443760/.49 = 2.33 Gflops / sec. total flop rate.
.69/.49 = 1.4x more time to run Proto stencil calculation.
.23/.327 = .7 i.e. a 30% speedup in residual calculation over previous pencil.
.558/.23 = 2.4x more time to compute the residual using Proto Stencil.

Also tried this leaving the multiplication by the coefs in — it made no difference.
Takeaways

• Going from pointwise operations to Pencil-based aggregate operations -> 20x-40x speedup. Can get within a 2X of the hand-coded version using the general-purpose stencil library.

• There is a significant difference between an outer loop over stencil locations and an inner pencil loop, and unrolling the stencil loop inside the pencil loop. Is there a way to do that in the general stencil apply code?

• Other than giving a crude cartoon for performance, we haven’t provided details of what causes the performance bottlenecks. Here are a couple of references: