Implementing Stencil Problems in Chapel: An Experience Report

Per Fuchs  Pieter Hijma  Clemens Grelck

Saturday 22 June 2019
CHIUW 2019
Context

- Teach Chapel in course
  - Programming Concurrent Systems (UvA)
  - Parallel Programming Practical (VU)
- Paper based on student report by Per Fuchs
- 4 versions
  - sequential
  - global view single locale
  - global view multi locale
  - local view multi locale
Context

System

- DAS-5
  - Dual socket, 8 cores, 2 hyperthreads
  - 48 Gbps InfiniBand
  - CentOS 7.4, 3.10.0 GNU/Linux

- Software
  - Chapel 1.19 (latest)
  - GCC 6.4.0 (latest version on DAS-5)
Motivation

- Over the years disappointing multi-locale performance
- Our attempt to get to the bottom of this
  - from users' perspective
- CHIUW as target in mind
- Can serve as explanation to (former) students
Conversation Users/Implementers

**Subject**: CHIUW: possible performance gotchas for your stencil computation

**To**: Clemens Grotz, Pieter Hjema, Per Fuchs

---

**Correspondents**

 Brad Chamberlain

**Date**

 27/04/2019, 04:07

**Location**

 Inbox

---

**From**: Brad Chamberlain

**Sent**: 29/04/2019, 13:34

**Subject**: CHIUW: possible performance gotchas for your stencil computation

---

**From**: Brad Chamberlain

**Sent**: 30/04/2019, 09:23

---

**From**: Brad Chamberlain

**Sent**: 02/05/2019, 19:49

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 02:40

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 15:36

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 16:18

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 17:07

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 17:45

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 17:47

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 21:02

---

**From**: Brad Chamberlain

**Sent**: 03/05/2019, 21:26

---

**From**: Brad Chamberlain

**Sent**: 04/05/2019, 09:39

---

**From**: Brad Chamberlain

**Sent**: 04/05/2019, 13:27

---

**From**: Brad Chamberlain

**Sent**: 27/04/2019, 04:07
```java
config const N = 8192;
config const M = 8192;
config const I = 500;
config const E = 0.01;

const HaloDomain: domain(2) = {0..N+1, 0..M+1};
const CylinderDomain: subdomain(HaloDomain) = {1..N, 1..M};

const LeftHalo: subdomain(HaloDomain) = {1..N, 0..0};
const RightHalo: subdomain(HaloDomain) = {1..N, M+1..M+1};
const UpperHalo: subdomain(HaloDomain) = {0..0, 0..M+1};
const LowerHalo: subdomain(HaloDomain) = {N+1..N+1, 0..M+1};

const LeftColumn: subdomain(HaloDomain) = {1..N, 1..1};
const RightColumn: subdomain(HaloDomain) = {1..N, M..M};
const UpperRow: subdomain(HaloDomain) = {1..1, 0..M+1};
const LowerRow: subdomain(HaloDomain) = {N..N, 0..M+1};

class Cylinder {
    var temperature : [HaloDomain] real;
}
```
Textbook implementation

```plaintext
config const N = 8192;
config const M = 8192;
config const I = 500;
config const E = 0.01;

const HaloDomain: domain(2) = {0..N+1, 0..M+1};
const CylinderDomain: subdomain(HaloDomain) = {1..N, 1..M};

const LeftHalo: subdomain(HaloDomain) = {1..N, 0..0};
const RightHalo: subdomain(HaloDomain) = {1..N, M+1..M+1};
const UpperHalo: subdomain(HaloDomain) = {0..0, 0..M+1};
const LowerHalo: subdomain(HaloDomain) = {N+1..N+1, 0..M+1};

const LeftColumn: subdomain(HaloDomain) = {1..N, 1..1};
const RightColumn: subdomain(HaloDomain) = {1..N, M..M};
const UpperRow: subdomain(HaloDomain) = {1..1, 0..M+1};
const LowerRow: subdomain(HaloDomain) = {N..N, 0..M+1};

class Cylinder {
    var temperature : [HaloDomain] real;
}
```
config const N = 8192;
config const M = 8192;
config const I = 500;
config const E = 0.01;

const HaloDomain: domain(2) = {0..N+1, 0..M+1};
const CylinderDomain: subdomain(HaloDomain) = {1..N, 1..M};

const LeftHalo: subdomain(HaloDomain) = {1..N, 0..0};
const RightHalo: subdomain(HaloDomain) = {1..N, M+1..M+1};
const UpperHalo: subdomain(HaloDomain) = {0..0, 0..M+1};
const LowerHalo: subdomain(HaloDomain) = {N+1..N+1, 0..M+1};

const LeftColumn: subdomain(HaloDomain) = {1..N, 1..1};
const RightColumn: subdomain(HaloDomain) = {1..N, M..M};
const UpperRow: subdomain(HaloDomain) = {1..1, 0..M+1};
const LowerRow: subdomain(HaloDomain) = {N..N, 0..M+1};

class Cylinder {
    var temperature : [HaloDomain] real;
}
for iteration in 1..I {
    forall (i,j) in zip(LeftHalo, RightColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i,j) in zip(RightHalo, LeftColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in CylinderDomain {
        var weight = conductivity[i, j];
        var remaining_weight = 1 - weight;
        dst.temperature[i, j] =
            weight * src.temperature[i, j] +
            // four direct neighbors
            remaining_weight * factor_direct_neighbors *
            (src.temperature[i-1, j] +
            src.temperature[i+1, j] +
            src.temperature[i, j-1] +
            src.temperature[i, j+1]);
    }
}

    max_difference = max reduce [ij in CylinderDomain] abs(dst.temperature[ij] - src.temperature[ij]);
    if max_difference < E break;
}

c src <=> dst;
for iteration in 1..I {
    forall (i,j) in zip(LeftHalo, RightColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i,j) in zip(RightHalo, LeftColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in CylinderDomain {
        var weight = conductivity[i, j];
        var remaining_weight = 1 - weight;
        dst.temperature[i, j] =
            weight * src.temperature[i, j] +
            // four direct neighbors
            remaining_weight * factor_direct_neighbors *
            (src.temperature[i-1, j] +
            src.temperature[i+1, j] +
            src.temperature[i, j-1] +
            src.temperature[i, j+1])
            // four diagonal neighbors
            remaining_weight * factor_diagonal_neighbors *
            (src.temperature[i-1, j-1] +
            src.temperature[i-1, j+1] +
            src.temperature[i+1, j-1] +
            src.temperature[i+1, j+1])
    }
    max_difference = max reduce [ij in CylinderDomain] abs(dst.temperature[ij] - src.temperature[ij]);
    if max_difference < E break;
    src <=> dst;
for iteration in 1..I {
    forall (i, j) in zip(LeftHalo, RightColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in zip(RightHalo, LeftColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in CylinderDomain {
        var weight = conductivity[i, j];
        var remaining_weight = 1 - weight;
        dst.temperature[i, j] =
            weight * src.temperature[i, j] +
            // four direct neighbors
            remaining_weight * factor_direct_neighbors *
            (src.temperature[i-1, j] +
            src.temperature[i+1, j] +
            src.temperature[i, j-1] +
            src.temperature[i, j+1]);
    }
    remaining_weight * factor_diagonal_neighbors *
    (src.temperature[i-1, j-1] +
    src.temperature[i-1, j+1] +
    src.temperature[i+1, j-1] +
    src.temperature[i+1, j+1]);
}
max_difference = max[reduce[i, j] in CylinderDomain]
abs(dst.temperature[i, j] - src.temperature[i, j]);
if max_difference < E break;
src <=> dst;
Vectorization by the C compiler

- vectorization has a large impact

Feedback from Chapel
- LLVM backend will give more control over vectorization
for iteration in 1..I {
    forall (i,j) in zip(LeftHalo, RightColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i,j) in zip(RightHalo, LeftColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in CylinderDomain {
        var weight = conductivity[i, j];
        var remaining_weight = 1 - weight;
        dst.temperature[i, j] =
            weight * src.temperature[i, j] +
            // four direct neighbors
            remaining_weight * factor_direct_neighbors *
            (src.temperature[i-1, j] +
             src.temperature[i+1, j] +
             src.temperature[i, j-1] +
             src.temperature[i, j+1]) +
            // four diagonal neighbors
            remaining_weight * factor_diagonal_neighbors *
            (src.temperature[i-1, j-1] +
             src.temperature[i-1, j+1] +
             src.temperature[i+1, j-1] +
             src.temperature[i+1, j+1]);
    }
    max_difference = max reduce [ij in CylinderDomain]
        abs(dst.temperature[ij] - src.temperature[ij]);
    if max_difference < E break;
    src <=> dst;
}
for iteration in 1..I {
    forall (i, j) in zip(LeftHalo, RightColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in zip(RightHalo, LeftColumn) {
        src.temperature[i] = src.temperature[j];
    }
    forall (i, j) in CylinderDomain {
        var weight = conductivity[i, j];
        var remaining_weight = 1 - weight;
        dst.temperature[i, j] =
            weight * src.temperature[i, j] +
            // four direct neighbors
            remaining_weight * factor_direct_neighbors * (src.temperature[i-1, j] +
            src.temperature[i, j-1] +
            src.temperature[i, j+1] +
            src.temperature[i+1, j]) +
            // four diagonal neighbors
            remaining_weight * factor_diagonal_neighbors *
            (src.temperature[i-1, j-1] +
            src.temperature[i-1, j+1] +
            src.temperature[i+1, j-1] +
            src.temperature[i+1, j+1]);
    }
    max_difference = max
    reduce [ij in CylinderDomain]
    abs(dst.temperature[ij] - src.temperature[ij]);
    if max_difference < E break;
    src <=> dst;
}
# Speedup vs. Threads

<table>
<thead>
<tr>
<th>Threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (s)</td>
<td>152.20</td>
<td>75.23</td>
<td>45.58</td>
<td>35.85</td>
<td>24.99</td>
</tr>
<tr>
<td>Speedup</td>
<td>0.99</td>
<td>2.01</td>
<td>3.32</td>
<td>4.22</td>
<td>6.06</td>
</tr>
<tr>
<td>BW STREAM (GB/s)</td>
<td>18.8</td>
<td>40.4</td>
<td>71.1</td>
<td>79.9</td>
<td>78.0</td>
</tr>
<tr>
<td>Required BW (GB/s)</td>
<td>22.9</td>
<td>46.4</td>
<td>76.6</td>
<td>97.3</td>
<td>139.6</td>
</tr>
</tbody>
</table>
use StencilDist;

var MyLocaleView = {0..numLocales-1, 0..0};

var MyLocales =
    reshape(Locales[0..numLocales-1], MyLocaleView);

const HaloDomain: domain(2) dmapped Stencil(
    boundingBox = {0..N+1, 0..M+1},
    targetLocales = MyLocales,
    fluff=(1,1)) = {0..N+1, 0..M+1};
```plaintext
for iteration in 1..I {
  ref s = src.temperature;
  ref d = dst.temperature;

  // exchange columns

  forall (i, j) in CylinderDomain {
    local {
      var weight = conductivity[i, j];
      var remaining_weight = 1 - weight;

      d[i, j] =
        weight * s[i, j] +
        remaining_weight * s[i + 1, j];

      // neighbor computations
    }
  }
  d.updateFluff();

  max_difference = max reduce [ij in CylinderDomain]
  abs(d[ij] - s[ij]);

  if max_difference < E break;
  src <=> dst;
}
for iteration in 1..I {
    ref s = src.temperature;
    ref d = dst.temperature;

    // exchange columns

    forall (i, j) in CylinderDomain {
        local {
            var weight = conductivity[i, j];
            var remaining_weight = 1 - weight;
            d[i, j] = weight * s[i, j] +
                        // neighbor computations
        }
    }
    d.updateFluff();

    max_difference = max reduce [ij in CylinderDomain]
                    abs(d[ij] - s[ij]);

    if max_difference < E break;
    src <=> dst;
}
Global View Multi Locale

<table>
<thead>
<tr>
<th>Nodes</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (s)</td>
<td>158.68</td>
<td>83.13</td>
<td>40.08</td>
<td>21.10</td>
<td>11.03</td>
</tr>
<tr>
<td>Speedup</td>
<td>0.16</td>
<td>0.30</td>
<td>0.62</td>
<td>1.18</td>
<td>2.27</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>0.16</td>
<td>0.15</td>
<td>0.16</td>
<td>0.15</td>
<td>0.14</td>
</tr>
</tbody>
</table>
Feedback from Chapel

Locality

- local blocks are deprecated
- localAccess expressions

NUMA awareness

- InfiniBand uses GASNet as communication layer
- GASNet is not NUMA aware
- Registering memory for the network is required
- Happens only on one NUMA-node
- Other NUMA-node suffers from this
for iteration in 1..I {
    ref s = src.temperature;
    ref d = dst.temperature;

    // exchange columns
    forall (i, j) in CylinderDomain {
        local {
            var weight = conductivity[i, j];
            var remaining_weight = 1 - weight;

            d[i, j] =
                weight * s[i, j] +
                // neighbor computations
        }
    }
    d.updateFluff();

    max_difference = max reduce [ij in CylinderDomain] abs(d[ij] - s[ij]);
    if max_difference < E break;
    src <=> dst;
}
for iteration in 1..I {
    ref s = src.temperature;
    ref d = dst.temperature;

    // exchange columns
    forall (i, j) in CylinderDomain {
        var weight = conductivity.localAccess[i, j];
        var remaining_weight = 1 - weight;
        d.localAccess[i, i] = remaining_weight;
        d.localAccess[i, j] = weight * s.localAccess[i, j] + remaining_weight;
        // neighbor computations
    }
    d.updateFluff();

    max_difference = max_reduce [ij in CylinderDomain]
        abs(d.localAccess[ij] - s.localAccess[ij]);

    if max_difference < E break;
    src <=> dst;
}
Performance difference

- factor 3.5 faster for 1 locale
- factor 2.4 faster for 16 locales
const Row : domain(1) = {0..M+1};

class Communicator {
    var firstRow: [Row] real;
    var lastRow: [Row] real;
}

var communicators: [LocaleSpace] Communicator;
```java
var globalIterations = 0;
var globalMaxDiff: real = min(real);

coforall l in Locales with
  (ref globalIterations, ref globalMaxDiff) {
  on l {
    const myCommunicator = new unmanaged Communicator();
    communicators[here.id] = myCommunicator;

    allLocalesBarrier.barrier();
    const isLast = here.id == LocaleSpace.last;
    const isFirst = here.id == LocaleSpace.first;

    const beforeCommunicator = if !isFirst
      then communicators[here.id - 1]
      else new unmanaged Communicator();
    const nextCommunicator = if !isLast
      then communicators[here.id + 1]
      else new unmanaged Communicator();
  }
```

const LocalCylinderDomain = CylinderDomain.localSubdomain();
const LocalHaloDomain = LocalCylinderDomain.expand(1, 1);

const LocalUpperRow = LocalHaloDomain.dim(1).first;
const LocalLowerRow = LocalHaloDomain.dim(1).last;
do {
    ref s = src.temperature;
    ref d = dst.temperature;

    local {
        local_max_difference = min(real);

        forall (i, j) in LocalCylinderDomain with
            (max reduce local_max_difference) {
            const weight = localConductivity[i, j];
            const remaining_weight = 1 - weight;
            const oldTemp = s[i, j];

            const newTemp = weight * oldTemp +
                // neighbor computations
                d[i, j] = newTemp;
            local_max_difference = max(local_max_difference,
                abs(new_temp - old_temp));
        }
        // exchange columns
    }

    beforeCommunicator.lastRow = d[LocalUpperRow + 1, ..];
    src <=> dst;

    // located on locale 0
    max_diffs[here.id] = local_max_difference;
    allLocalesBarrier.barrier();
    if (!isLast) {
        forall col in Row {
            d[LocalLowerRow, col] = myCommunicator.lastRow[col];
        }
    }
    if (!isFirst) {
        forall col in Row {
            d[LocalUpperRow, col] =
        }
    }
    local_max_difference = max reduce max_diffs;
    local_iteration += 1;
    while (local_max_difference > E && local_iteration < I);
Local View Multi Locale

<table>
<thead>
<tr>
<th>Nodes</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (s)</td>
<td>47.15</td>
<td>24.74</td>
<td>14.55</td>
<td>7.62</td>
<td>4.33</td>
</tr>
<tr>
<td>Speedup</td>
<td>0.53</td>
<td>1.01</td>
<td>1.72</td>
<td>3.28</td>
<td>5.78</td>
</tr>
<tr>
<td>Efficiency</td>
<td>0.53</td>
<td>0.51</td>
<td>0.43</td>
<td>0.41</td>
<td>0.36</td>
</tr>
</tbody>
</table>
Same issues

- locality control
- NUMA awareness of the communication layer
Performance difference

- factor 1.46 faster for 1 locale
- factor 1.36 faster for 16 locales
Conclusions

- We can explain the disappointing performance
- We cannot achieve high performance on our system
  - NUMA awareness of communication layer remains a problem
  - new OpenFabrics layer is promising
- Chapel is a very powerful and expressive language
for iteration in 1..I {
    ref s = src.temperature;
    ref d = dst.temperature;

    // exchange columns
    forall (i, j) in CylinderDomain {
        local {
            var weight = conductivity[i, j];
            var remaining_weight = 1 - weight;

            d[i, j] = weight * s[i, j] +
            // neighbor computations
        }
    }
    d.updateFluff();

    max_difference = max reduce [ij in CylinderDomain]
            abs(d[ij] - s[ij]);
    if max_difference < E break;
    src => dst;
}
All Attempts Multi Locale Global View (speculative)
All Attempts Multi Locale Local View (speculative)

![Graph showing speedup vs. number of nodes for different locale settings: Ideal, par-local-multi, par-local-multi-localaccess, par-local-multi-cray. The graph illustrates the performance improvement as the number of nodes increases.]