### Pragma based parallelization - Trading hardware efficiency for ease of use?

**Tobias Kenter** kenter@upb.de

**Pragma based parallelization**

<table>
<thead>
<tr>
<th>Why?</th>
<th>productive development</th>
<th>single source principle</th>
<th>transparent calls and synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>How?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
#pragma cny begin_coproc
for (i = 0; i < size; i++) {
  y[i] = a * x[i] + y[i];
}
#pragma cny end_coproc
```

<table>
<thead>
<tr>
<th>Questions?</th>
<th>suitable for FPGAs?</th>
<th>efficiency tradeoffs?</th>
<th>coding effort?</th>
</tr>
</thead>
</table>

**Case Study Stereo Matching**

![Image](image-url)

[Mei et al. On building an accurate stereo matching system on graphics hardware. In Proc. ICCV Workshop on GPU in Computer Vision Applications (GPUCV, 2011)](image-url)

**Convey HC-1 Platform**

- **Coprocessor**
  - 1 Application Engine Hub: interaction with host execution of scalar code
  - 4 Application Engines: Xilinx Virtex-5LX330 for user logic
  - 8 memory controllers: aggregated bandwidth 80 GB/s

- **Memory**
  - shared address space
  - physically separate memory

**Vector Personality**

Vector coprocessor in AEs:
- 1024 element vectors
- single precision floating point
- indexed loads and stores
- masked stores

Compiler support:
- automatic vectorization
- various pragmas:
  - memory, dependencies, array dimensions

**Performance**

(all tests on HC-1 with scatter/gather RAM)
- Peak compute 38.4 GFLOPS (synthetic microbenchmark without memory access in innermost loop)
- Saxpy compute 7.36 GFLOPS
- Scanline compute 7.16 GFLOPS
- Saxpy bandwidth 44.2 GB/s
- Scanline bandwidth 8.93 GB/s

### Cost Initialization

How likely does a pixel in the right image fit the corresponding pixel in the left image? Compute this value for all possible disparities (distances in x-positions between possibly corresponding pixels).

- **Census cost:** bit level pattern
- **Absolute difference (AD)**

Scaled sum of both metrics

<table>
<thead>
<tr>
<th>Census</th>
<th>Hamming Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Costs</td>
<td></td>
</tr>
</tbody>
</table>

#### Cost Aggregation

Filter the noisy initial costs: sum up cost of neighboring and similar pixels, because they are likely to have similar disparity values.

Three compute patterns:
- Compute support regions (iterative)
- Accumulate line / column values
- Compute sums for support regions

```c
for(d=0; d<max_disparity; d++)
for(x=0; x<width; x++)
for(y=0; y<height; y++)
  ri = right[y][x]; //scalar expansion
  le = left[Y][x];
  costs[1][d][y][x+ri] = costs[1][d][y][x+le];
```

Intermediate results before and after Cost Aggregation

**Scanline Optimization**

Minimize energy of cost + disparity gradients along horizontal and vertical scanlines.

- **Two aspects:**
  - calculate penalties depending on color difference values
  - minimize dynamic programming scheme

```c
for(d=0; d<max_disparity; d++)
for(x=0; x<width; x++)
for(y=0; y<height; y++)
  if (cost[d-1][y][x]+pl < min_cost)
    min_cost = cost[d-1][y][x]+pl;
```

### Convey Vector Personality

- **significant speedups** possible
- **tradeoffs** in performance and flexibility
- full support for **masked operations** missing
- memory speed limits performance
- suitable for **different compute phases / patterns**

### Convey Vectorizing Compiler

- **transparent accelerator calls** support productivity
- **significant effort** needed for any vectorization
- vectorization of **only inner loops** limits performance
- **missing reuse** limits performance
- missing support for **masked operations** limits applicability

---

**Conclusions**

Tobias Kenter, Henning Schmitz, Christian Plessl
Paderborn Center for Parallel Computing, University of Paderborn, Germany
kenter@upb.de, mod@upb.de, christian.plessl@upb.de