Reduced precision applicability and trade-offs for SLAM algorithms

Oscar Palomar, Andy Nisbet, John Mawer, Graham Riley, Mikel Lujan

Advanced Processor Technologies (APT)
University of Manchester, UK

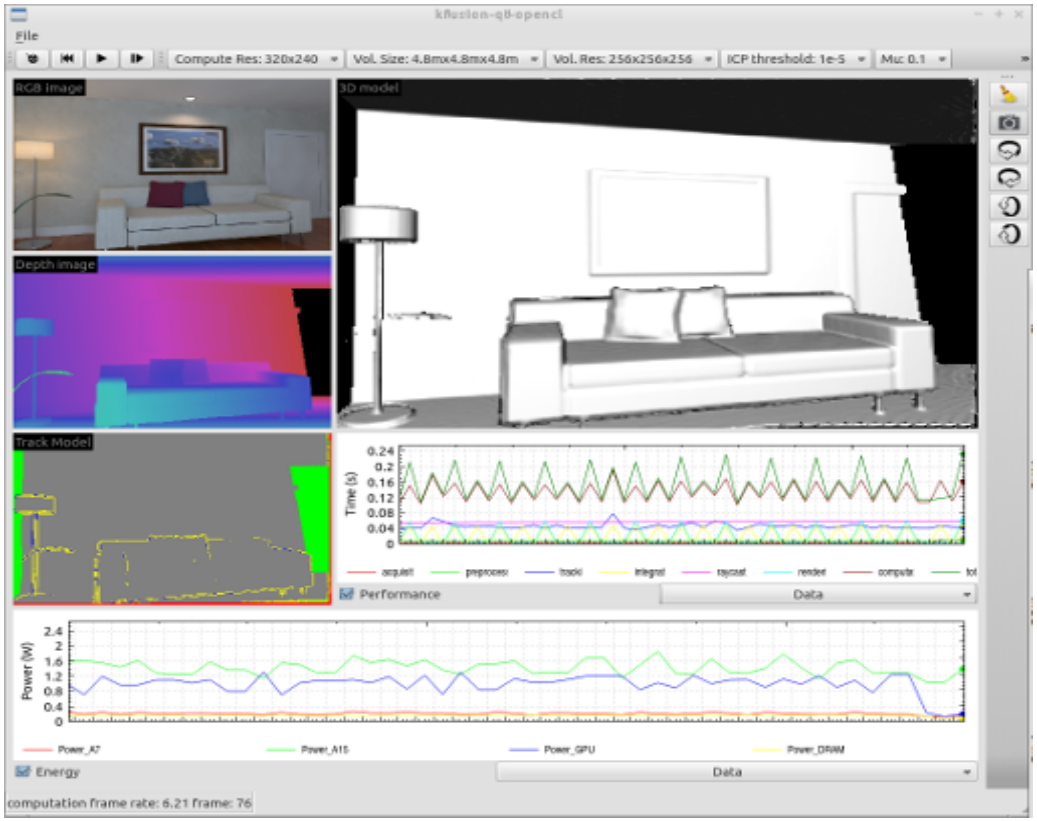
oscar.palomar@manchester.ac.uk

3rd Workshop On Approximate Computing (WAPCO 2017)

25/01/2017, Stockholm

SLAM

- SLAM: Simultaneous Localisation And Mapping
  - Determine the pose (position, orientation) of a device with camera(s) and map the environment
  - Stereo vs. monocular + depth camera (RGB-D)
  - Dense vs. sparse
  - KinnectFusion, ORBSlam2, ElasticFusion, LSDSlam,…
- SLAMBench from the PAMELA project
  - http://apt.cs.manchester.ac.uk/projects/PAMELA/tools/SLAMBench/index.html#download
  - Clean, efficient implementation of KinnectFusion
  - Multiple variants: C++, OpenMP, CUDA, OpenCL
  - Report performance, power, accuracy (vs. known ground truth): Absolute Trajectory Error



Approximate computing opportunities in SLAM

- Accuracy metric, known ground truth (when using the standard, synthetic ICL data sets)
  - Straightforward to determine the impact of approximation
- Already include algorithmic parameters that affect accuracy
- Input is a (noisy) image stream
  - Algorithms must be robust to errors there
- “Delta” between frames is typically small
  - Some selected as keyframes. Compute these only with the highest accuracy?
  - Beware of range if reducing precision though…

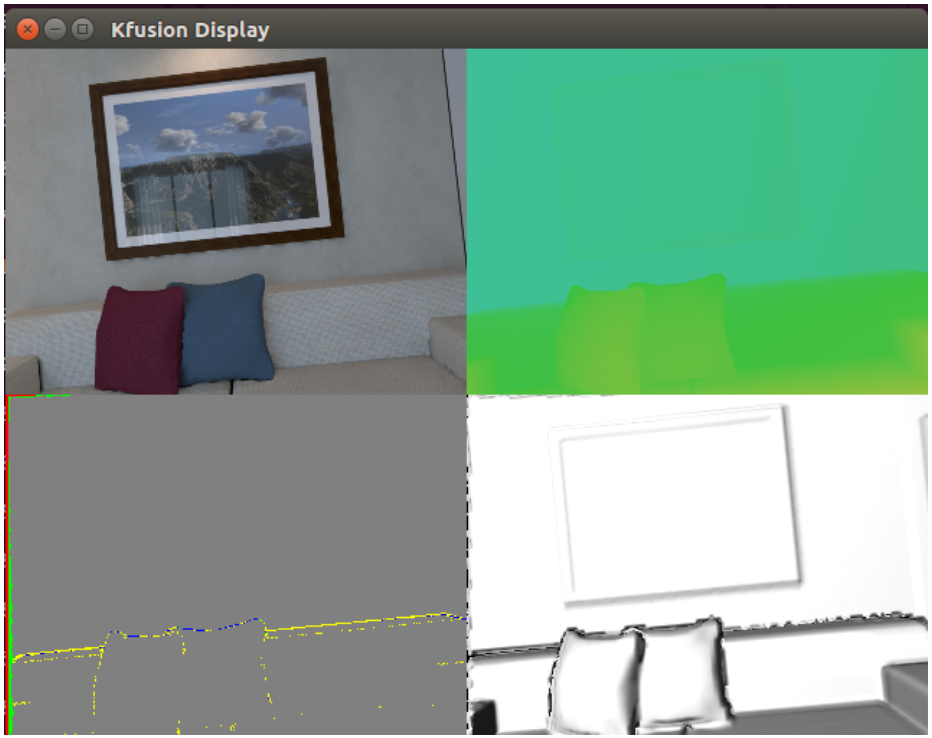
Reduced precision

- Use 16-bit floating (half) in SLAMBench on ARM processors
  - IEEE standard, ARM support (CUDA too)
    - __fp16 type in gcc (limited, e.g. cannot be a function parameter or return type)
    - ARM instructions for load/store only by now
- First attempt, use it everywhere
  - Failed, the algorithm is unable to track
  - Some variables need 32-bit
- Tried 16-bit as default and selectively use 32-bit where necessary
  - Also failed
  - Long, tedious debugging and tracing efforts useless
- Now my default is 32-bit and selectively use 16-bit
  - It works :)
  - Incrementally expand use of 16-bit
  - Easier to debug (there is a correct baseline)

FlexiblePoint Library

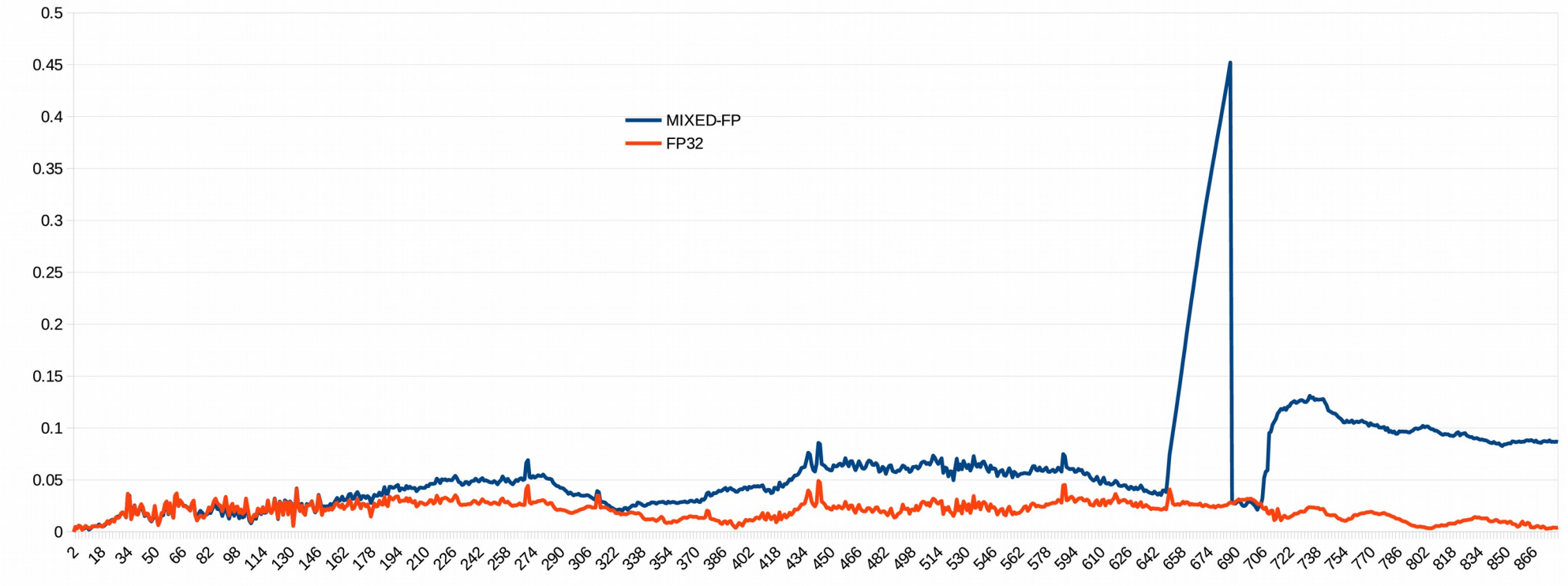
- flexpoint C++ class, floating point variables that can store data in 32-bit or 16-bit format (and change between them)
- Methods to set the default width and per variable width
- Overload all operators to operate with flexpoint variables set to 16-bit, 32-bit or a mix of both
- Implicit cast from/to int/long/char/float…
  - Float for constants, or return values/parameters of library function calls
  - Worked fined for the majority of the code, needed explicit casting in a few places
- Generate statistics, traces, track ranges, etc...
- Developed to be able to experiment with mixed precision easily and to facilitate changing the data width for design space exploration (DSE)
  - Significant slowdowns

Overview of results



configuration	ATE Max	Mean	Total
FP64	0.049	0.020	18.051
FP32	0.049	0.020	18.067
MIXED (72% FP16)	0.452	0.062	54.513

Absolute Trajectory Error



The integrate kernel

```
void integrateKernel(Volume vol, const float* depth, uint2 depthSize,
  const Matrix4 invTrack, const Matrix4 K, const float mu, const float maxweight) {

  float3 delta = rotate(invTrack, make_float3(0, 0, vol.dim.z / vol.size.z));
  const float3 cameraDelta = rotate(K, delta);
  unsigned int y;

  for (y = 0; y < vol.size.y; y++) {
    for (unsigned int x = 0; x < vol.size.x; x++) {
      uint3 pix = make_uint3(x, y, 0); //pix.x = x;pix.y = y;
      float3 pos = invTrack * vol.pos(pix);
      float3 cameraX = K * pos;
      for (pix.z = 0; pix.z < vol.size.z;
        ++pix.z, pos += delta, cameraX += cameraDelta) {
        if (pos.z < 0.0001f) continue; // some near plane constraint
        const float2 pixel = make_float2(cameraX.x / cameraX.z + 0.5f,
          cameraX.y / cameraX.z + 0.5f);
        if (pixel.x < 0 || pixel.x > depthSize.x - 1 || pixel.y < 0
            || pixel.y > depthSize.y - 1) continue;
        const uint2 px = make_uint2(pixel.x, pixel.y);
        if (depth[px.x + px.y * depthSize.x] == 0)
          continue;
        const float diff =
          (depth[px.x + px.y * depthSize.x] - cameraX.z)
          * std::sqrt(1 + sq(pos.x / pos.z) + sq(pos.y / pos.z));
        if (diff > -mu) {
          const float sdf = fminf(1.f, diff / mu);
          float2 data = vol[pix];
          data.x = clamp((data.y * data.x + sdf) / (data.y + 1), -1.f,
            1.f);
          data.y = fminf(data.y + 1, maxweight);
          vol.set(pix, data);
        }
      }
    }
  }
}
```

configuration	ATE Max	Mean	Total	Not tracked
FP16	3.505	1.904	1675	708
VOL-FP32	3.101	1.397	1229	385
CAM-Z-FP32	0.712	0.346	304	41
CAM-FP32	0.057	0.026	23	0
CAM-VOL-FP32	0.056	0.026	23	0
FP32	0.049	0.020	18	0

Conclusions and future work

- SLAMBench can benefit a lot from reduced precision
  - But it is “fragile”
- Will need more DSE
  - Automated for larger design space
    - Width of multiple (each) variables, input data set, algorithmic parameters...
    - Other SLAM algorithms
- Performance, power, energy evaluation
  - Efficient implementation from selected points
  - SLAMbench CUDA implementation of Kfusion
- Dynamic tuning, keyframes vs non-keyframes,…

Reduced precision applicability and trade-offs for SLAM algorithms

Oscar Palomar, Andy Nisbet, John Mawer, Graham Riley, Mikel Lujan

Advanced Processor Technologies (APT)
University of Manchester, UK

oscar.palomar@manchester.ac.uk

3rd Workshop On Approximate Computing (WAPCO 2017)

25/01/2017, Stockholm