A Graphics Processing Unit Implementation of the Particle Filter

AUTOMATIC CONTROL
REGLERTEKNIK
LINKÖPINGS UNIVERSITET

Gustaf Hendeby, Jeroen D. Hol, Rickard Karlsson, Fredrik Gustafsson

({hendeby, hol, rickard, fredrik}@isy.liu.se)

AUTOMATIC CONTROL
REGLERTEKNIK
LINKÖPINGS UNIVERSITET

Graphical Processing Unit (GPU):

- Is found in almost any modern computer
- Features parallel programmable (SIMD) hardware
- Is an alternative to parallel multi-core, FPGA, or cluster programming for testing and algorithm development

Particle Filter (PF):

- Provides an approximate solution to the general nonlinear filtering problem
- Is *almost parallel* in its structure
- Gains a lot from an efficient parallel implementation

Particle Filter

A particle filter (PF) is used to estimate the state vector x_t from measurements y_t, related to each other via the nonlinear model,

$$x_{t+1} = f(x_t, w_t)$$
$$y_t = h(x_t) + e_t,$$

where w_t and e_t are process and measurement noise, respectively, and have known, but not necessarily Gaussian, distributions; p_w and p_e.

The particle filter is a good alternative when it comes to filtering of nonlinear and non-Gaussian models, however it is computationally quite expensive. Methods to *speed up the filtering* are hence vital.

PF: A Parallel Algorithm

Initialization:
Generate N particles $\{x_0^{(i)}\}_{i=1}^N \sim p(x_0)$

Measurement update:
Compute the particle weights
$\omega_t^{(i)} = p(y_t|x_t^{(i)}) \,/\, \sum_j p(y_t|x_t^{(j)})$

The weight of all particles can be computed independently in the *measurement update*, and hence in *constant time* when fully parallelized.

Resample:

1. Generate N uniform random numbers $\{u_t^{(i)}\}_{i=1}^N \sim \mathcal{U}(0,1)$.
2. Compute the cumulative weights: $c_t^{(i)} = \sum_{j=1}^i \omega_t^{(j)}$.
3. Generate N new particles using $u_t^{(i)}$ and $c_t^{(i)}$: $\{x_t^{(i\star)}\}_{i=1}^N$ where $\Pr(x_t^{(i\star)} = x_t^{(j)}) = \omega_t^j$.

During *resampling* all particles interact, making *parallelization difficult* but to a certain degree still possible.

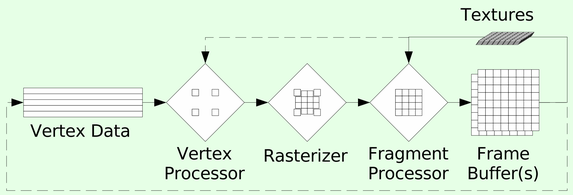
Time update:

1. Generate process noise $\{w_t^{(i)}\}_{i=1}^N \sim p_w(w_t)$.
2. Simulate new particles $x_{t+1}^{(i)} = f(x_t^{(i\star)}, w_t^{(i)})$.

During the *time-update step* every particle is handled independently, which *parallelizes perfectly*.

GPU Basics

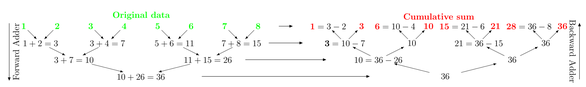
- A GPU can be found in most computers.
- Workflow determined by the graphics pipeline:



- Two sets of programmable units with parallel SIMD architecture:
  - vertex shader (used to handle the transformation of triangles).
  - fragment shader (program to calculate potential pixels).
- Typical workflow:
  1. Program the fragment shader with the desired operation.
  2. Send the data to the GPU in the form of a texture.
  3. Draw a rectangle of suitable size on the screen to start the calculations.
  4. Read back the resulting texture to the CPU.

PF GPU

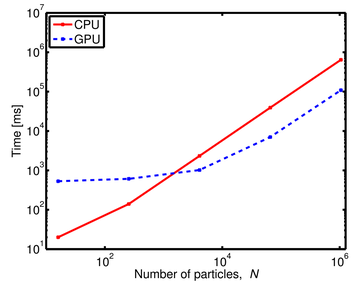
The seemingly nonparallel resampling step can be implemented using some more operations that can be parallelized to a higher degree. Below is an example of how to obtain the cumulative sum of a set of numbers. Sequentially it demands $\mathcal{O}(N)$ sequential operations, which can be cut down to $\mathcal{O}(\log N)$ in time using $\mathcal{O}(N \log N)$ operations in total. This is an essential part of making the resampling parallel.



Parallelized cumulative sum operation.

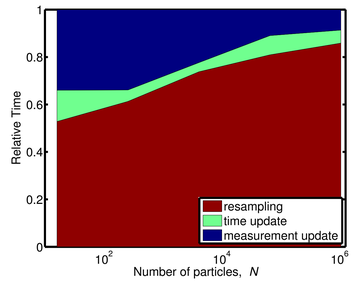
Result

- Implemented successfully on an NVIDIA 7900GTX (8/24 vertex/fragment pipelines).



Time needed to run particle filter.

- The PF GPU implementation has a nice time complexity compared to a CPU implementation. (The gain is lost as the number of particles increases, and for few particles the overhead of interacting with the GPU is significant. Whether the GPU implementation is an improvement depends on the ratio between overhead and number of pipelines.)



Proportional time for the different steps in the parallel algorithm.

- In simulations the resampling is shown to take the most time and dominate for many particles (as expected).

Conclusion

- The first complete and general particle filter implemented on a GPU.
- Its time complexity is shown to be encouraging.