graphics hardware 06

Pseudorandom Number Generation on the GPU

Myles Sussman, William Crutchfield, Matthew Papakipos

PEAKSTREAM

Outline

- Motivation & Constraints (Why and What)
- Review : CPU-based Linear RNGs
  - Parallelization strategy
- Why Linear RNGs are impractical on GPUs (now)
- Nonlinear RNGs
- Gotchas
- Performance on real GPUs
- Conclusions & Suggestions for the Hardware

Motivation & Constraints

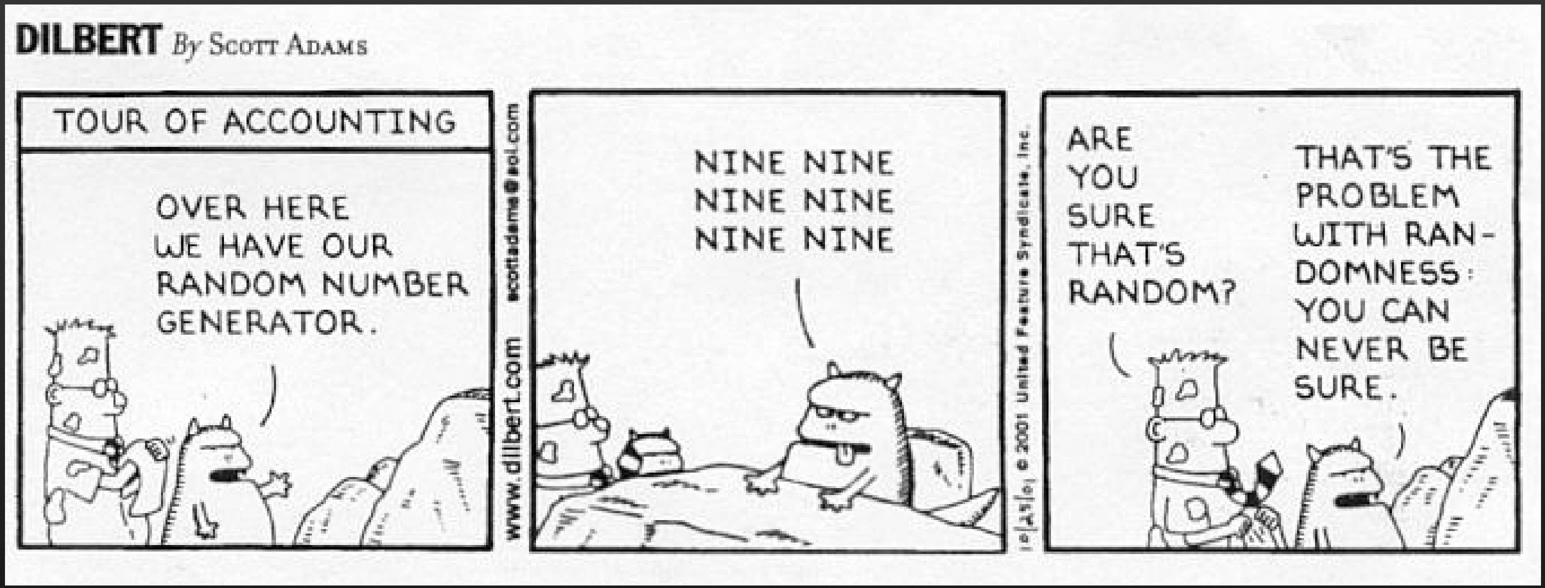
Why?

- Use GPU for Monte Carlo integration
- Ideal for GPGPU : compute a lot, output a little
  - Mean, median; uncertainty ~ $O(1/\sqrt{N})$
  - Generate random numbers on CPU implies lots of traffic

What?

- Don’t reinvent the RNG wheel!
  - Lots of existing theory on RNGs
  - “Industry standards” : MKL (Intel), ACML (AMD), others

Randomness



Diehard and TestU01 : is it random enough?

- Like repeated poker games
- Ensure the house isn’t cheating (p-value)

Linear RNGs

- Modulus m, multiplier a
- Sequence, period is m
- Output u in [0,1)
- Many types : LCG, MCG, MRG
- Combined generators have larger period (e.g. m_1 x m_2)
- Data dependency : "seed" or previous value

$$x_n = (a x_{n-1} + c) \bmod m$$

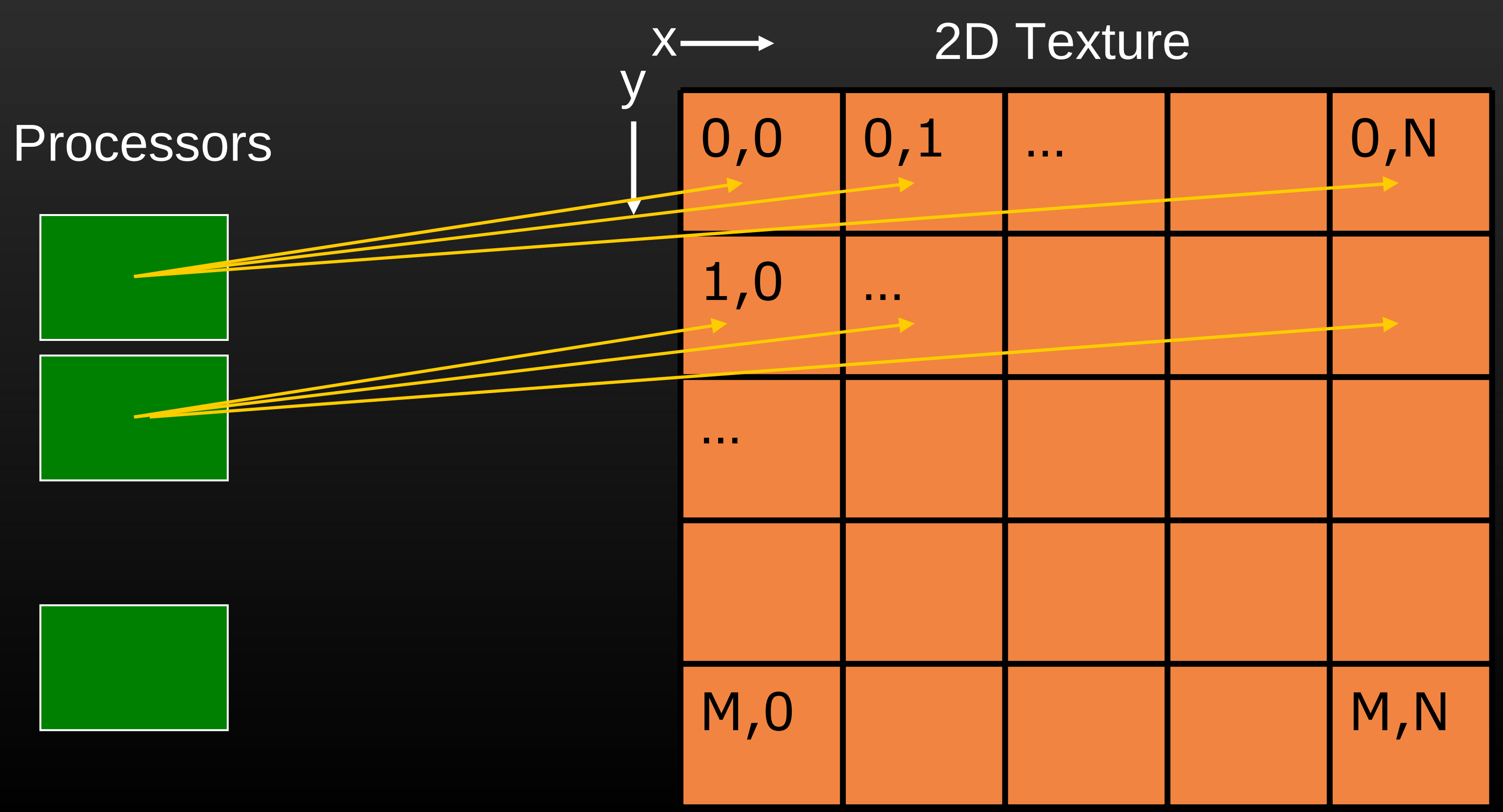
$$u_n = x_n / m$$

$$v_n = (u_{1n} + u_{2n}) \bmod 1$$

Parallelizing

Each pixel is a separate (virtual) thread

- Required : ***independent sequence at each pixel***



Parallelizing

Each pixel is a separate (virtual) thread

- Required : ***independent sequence at each pixel***

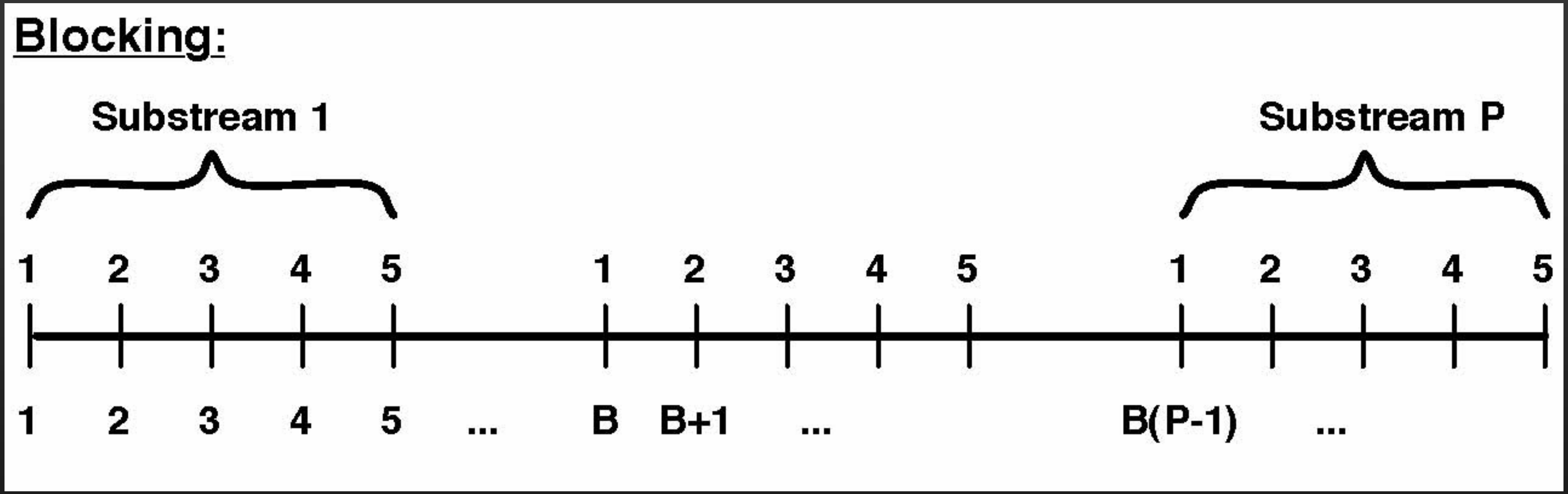
How to achieve independence:

- Different methods : Wichmann-Hill family (273 methods)
- One long sequence with each pixel assigned a different “block” : MRG32k3a

PEAKSTREAM

graphics hardware 06

Blocking



- Each pixel (substream) outputs 1 block from long sequence
- Easy to get burned! Linear RNG = long-range correlations
- MRG32k3a painstakingly optimized, minimizes correlations

How Much Seed Data?

Each thread can only write 16 floats

- At least one is your result
- Others are needed to update the seed
  - MRG32k3a = 6 doubles = 12 floats, leaves 4 results
  - 4096 x 4096 x 4 buffer of results = 192 MB of seed!
- Seed update from CPU = slow
- What about Wichmann-Hill ?
  - 273 methods = each needs to write 240K results!
- Linear RNG isn't practical today

Nonlinear RNGs

$$x_n = \overline{a(n + n_0 + c)} \bmod m$$

Explicit Inverse Congruential Generator

- No data dependency, directly compute
- Sequence, period is m
- May be combined, period is m_1 x m_2
- Fewer correlation "troubles"
- Compute cost ~ O(log(m)) = more expensive
- But GPUs are faster …

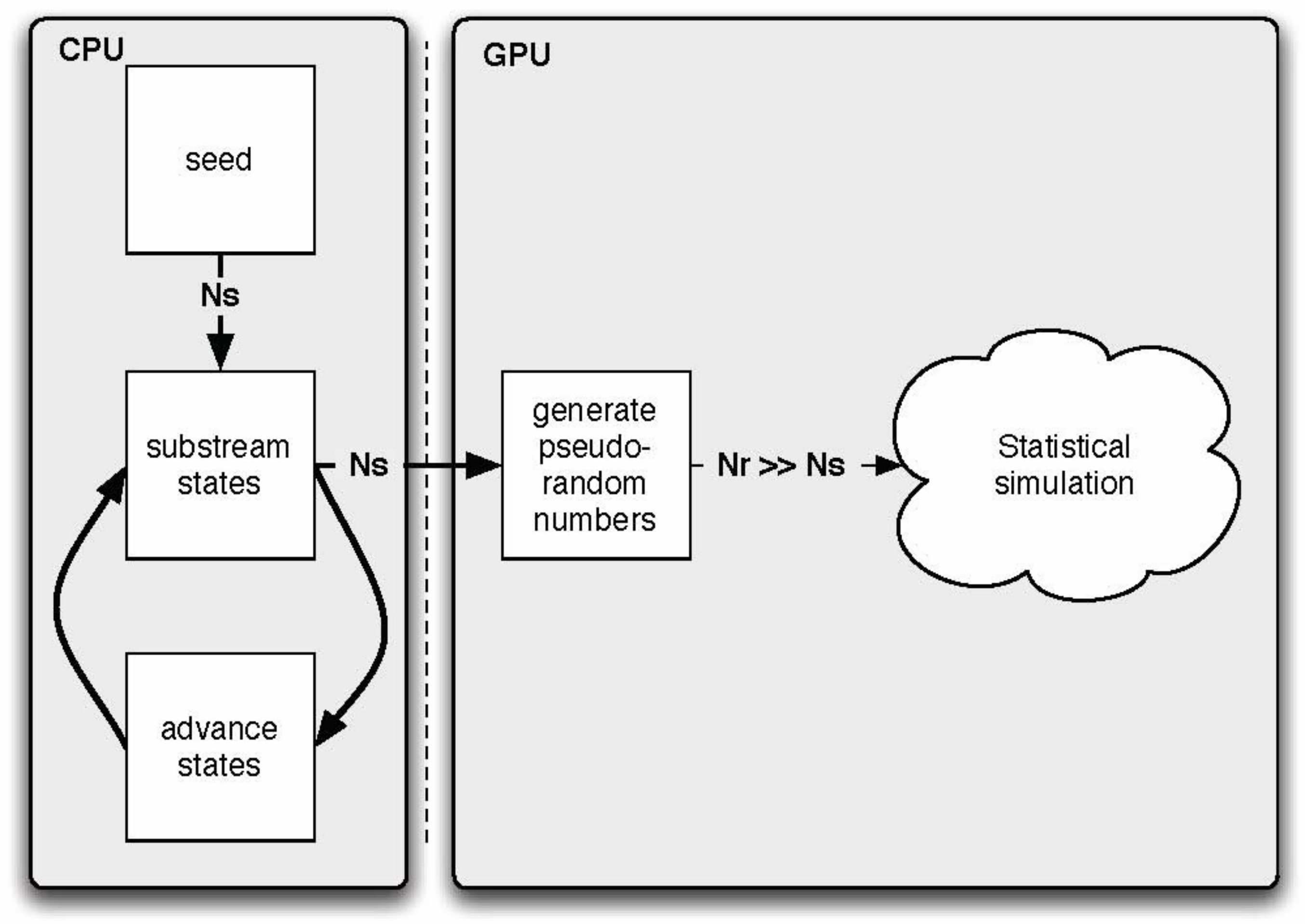
Parallelizing Made Simple

Pixel at texture coordinate (x,y)

$$x_n = \overline{a(n + n_0 + (x + 4096y)B)} \bmod m$$

- 4096 x 4096 independent blocks of length B
- Floating point math = m is 24 bits
- Tricks must be played to keep within 24 bits
- Seed data $n+n_0$ is the same for all pixels!
  - Can be managed on CPU or GPU or both (~100 bytes)

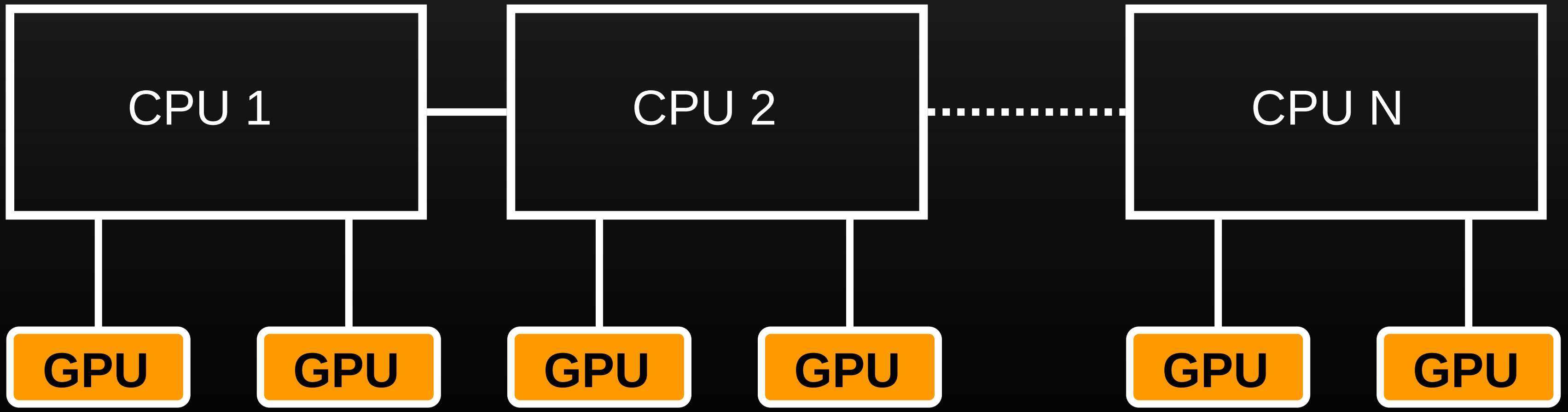
Managing Seed Data



"Ultimate" Architecture?

Blocking = independent substreams

- Seeds for GPUs are advanced by "cluster sub-block size"
- Many cluster architectures possible



Gotchas

Some things are different …

- Integer division is inexact
  - N/N doesn't always equal 1
  - Remainder can be off by ±1 (= error in mod)
  - Need special tricks (see the paper)
- Floating point math = 24 bits
  - MRG32k3a designed for 53 bits (doubles) : requires three floats to store intermediates
  - Nonlinear RNG : combine three 24-bit generators for long period

PEAKSTREAM

graphics hardware 06

Performance of RNGs

RNG type	Usable sequence length at each pixel	ATI Radeon X1900 500 MHz	Intel Xeon 3.6 GHz
Nonlinear	$\sim 2^{45}$	45 million/sec	0.3
MRG32k3a	$>2^{46}$	110 kernel only	110 MKL
Wichmann-Hill	$>2^{32}$	823 kernel only	79 MKL

Unlimited Outputs Per Thread

- Wichmann-Hill ops are 10x faster vs CPU
  - But we need >240K outputs per thread
- For MRG32k3a ops are same speed vs CPU
  - Anticipate large speedup with ints (DirectX 10)
    - (or if we have doubles)
  - But we need many more outputs per thread

Performance of RNGs

RNG type	Usable sequence length at each pixel	ATI Radeon X1900 500 MHz	Intel Xeon 3.6 GHz
Nonlinear	$\sim 2^{45}$	45 million/sec	0.3
MRG32k3a	$>2^{46}$	110 kernel only	110 MKL
Wichmann-Hill	$>2^{32}$	820 kernel only	79 MKL
New nonlinear	$\sim 2^{46}$	1300	3.3

Conclusions & Suggestions

Can do RNGs / Monte Carlo on GPU !

- Nonlinear RNGs : A good solution *today*
- Linear RNGs would be better *if…*

Desired hardware features :

- Unlimited (or many more) outputs per thread
- Integers (DirectX 10) & doubles
- More instructions in each shader program

myles@peakstreaminc.com

HLSL sample : Accurate mod

```
float4 mod_div(float4 a, float4 b, out float4 d) {
    d = floor(a/b);
    float4 r = a - d*b;
    // handle case where division off by -1 ulp
    d = (r<0) ? d-1 : d;
    r = (r<0) ? r+b : r;
    // handle case where division off by +1 ulp
    d = (r<b) ?   d : d+1;
    r = (r<b) ?   r : r-b;
    return r;
}
```

HLSL sample : Pixel Shader

```
/* seed data for all components, used by ceicg_cpu_4 */
sampler seed_data;

/* generate 4 random numbers at each pixel position */
float4 ceicg_gpu_4( float2 pixel_pos ) {
   /* depends only on pixel position and seed data */
}

struct PS_OUTPUT {
    float4 color0 : COLOR0;
};

/* main pixel shader program for nonlinear RNG */
PS_OUTPUT ps_main(float2 pos : VPOS) {
    PS_OUTPUT po;
    po.color0 = ceicg_gpu_4(pos);
    return po;
}
```