Tennessee Advanced Computing Laboratory
National Science Foundation Grant, CHE-0625598

PROGRAMMABILITY: DESIGN COSTS AND PAYOFFS USING AMD GPU STREAMING LANGUAGES AND TRADITIONAL MULTI-CORE LIBRARIES

7/30/09

Rick Weber, Robert Harrison, Greg Peterson

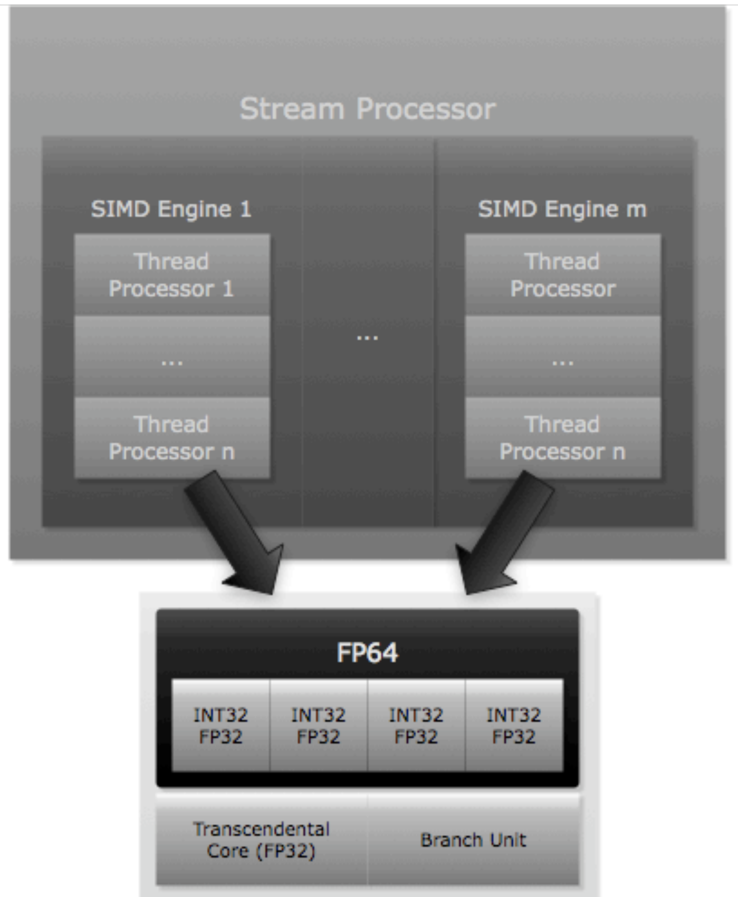
Multi-core Processors

- Put multiple CPUs on a single chip
- Multi-core becoming the norm in computing
  - Helps overcome power wall
  - Sometimes helps overcome memory wall
  - More efficient use of transistors than complex branch prediction, score boarding, etc.
- Usually requires explicit software support to reap benefits (in a single application)
  - Unlike many CPU performance features, speedups don't come for "free"
  - Requires developers to exploit thread level parallelism (usually explicitly)
  - OpenMP, pthreads, Intel Threading Blocks

General Purpose Graphics Processing Unit (GPGPU)

- Traditionally used for gaming and CAD
  - Fixed pipeline replaced with programmable shader units
  - Lots of simple processors for computing pixels in parallel
  - Large memory bandwidth for sampling textures
- Now being used in general purpose computing
  - CUDA, Brook+, CAL, OpenCL all target GPUs
  - Can achieve exploit massive parallelism
  - Double precision now supported on some cards
    - Firestream series
    - Tesla 1060

ATI GPU architecture



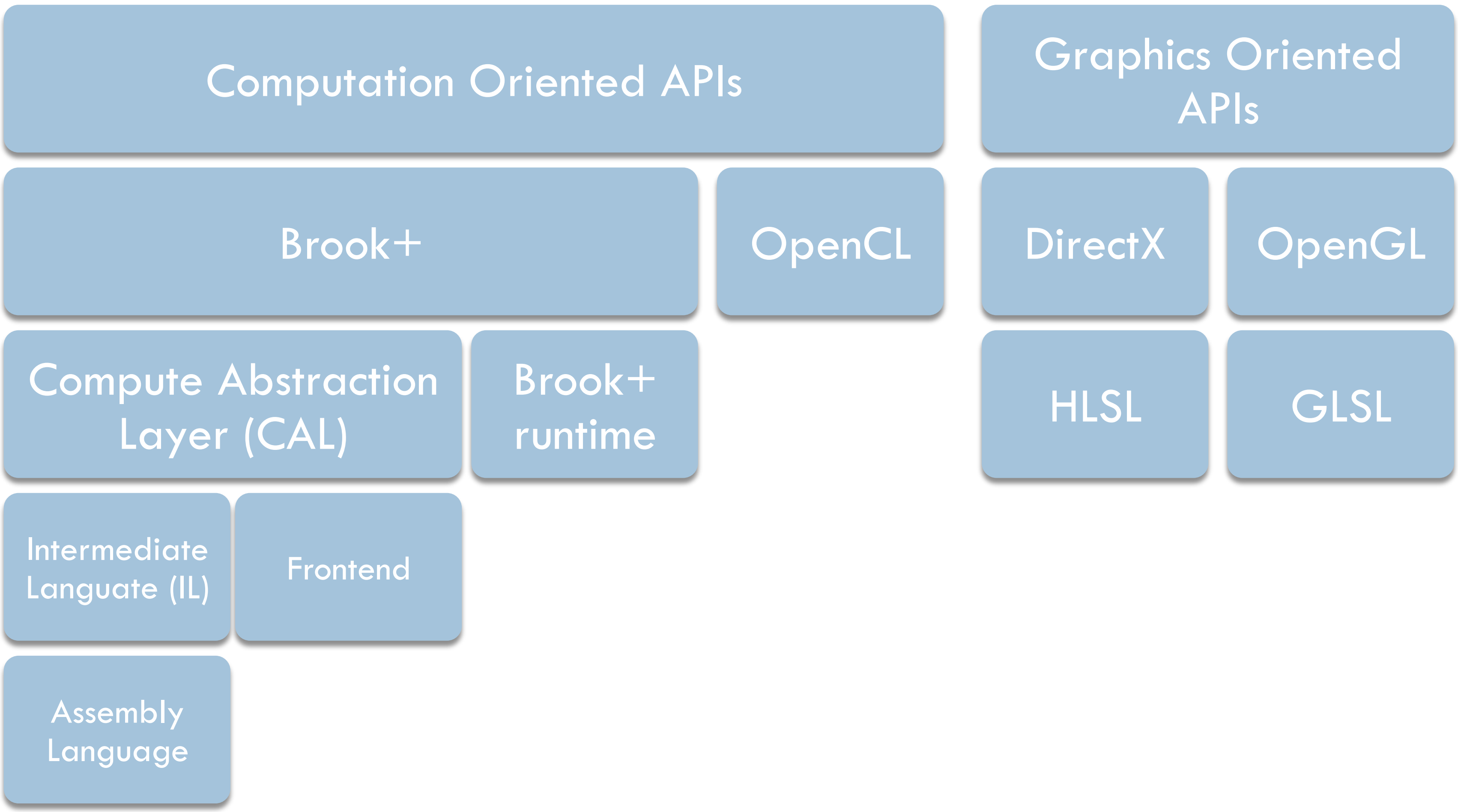
ATI GPU Architecture

- Stream processor bundles SIMD Engines
- SIMD Engine
  - Controls thread processor execution
  - All thread processors within SIMD engine execute same instruction
  - Analogous to Nvidia's SIMT Multiprocessors
  - Very long instruction word (VLIW)
- Thread processors
  - Analogous to Nvidia's thread processos
  - 5 floating point units (stream cores) per processor

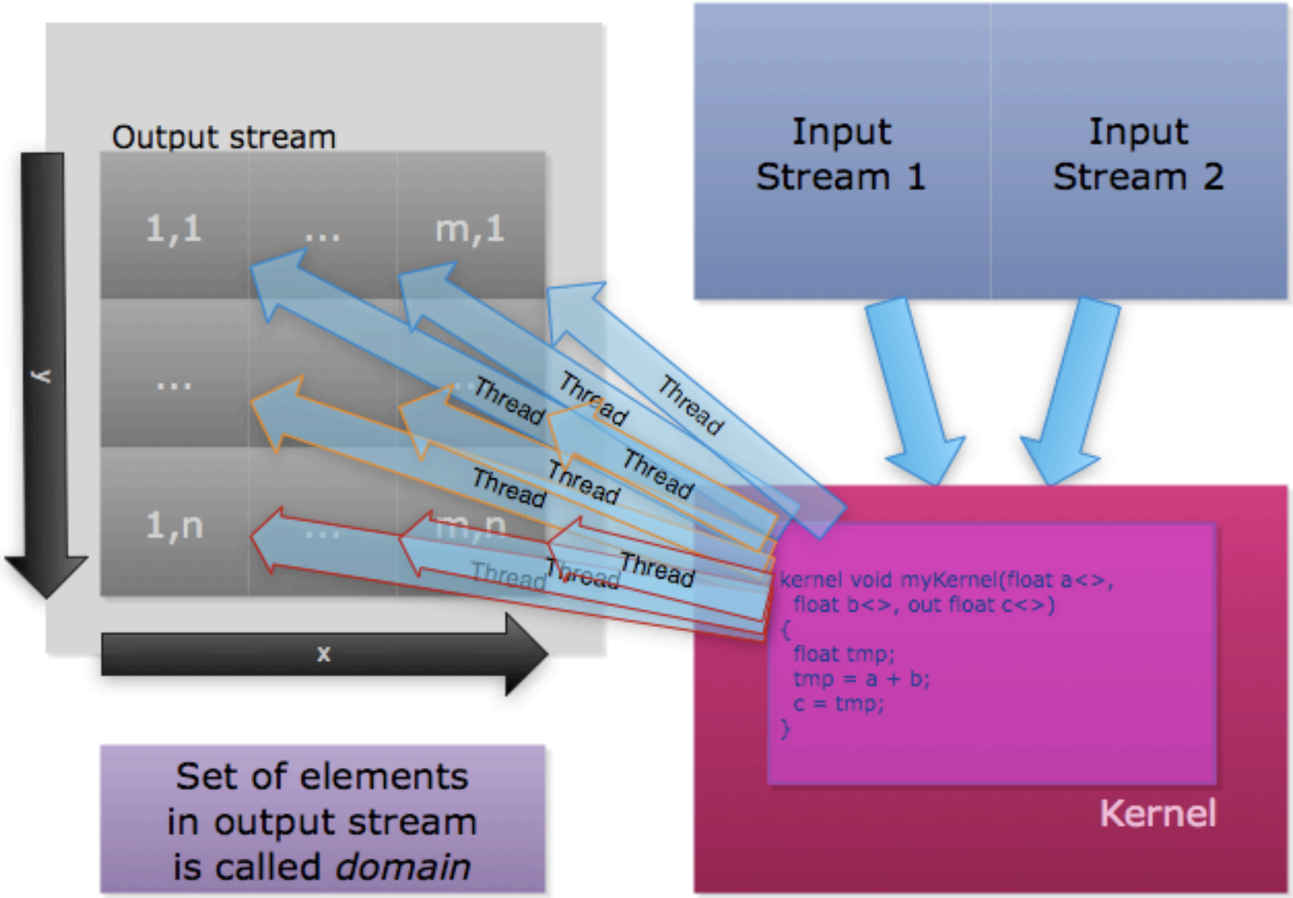
Firestream 9170

- 320 Stream cores
- ~500Gflops/s peak (single precision)
- ~100Gflops/s peak (double precision)
  - 4 SP units are fused to perform one double operation
  - T-unit is not used in double
- 51.2GB/s memory bandwidth

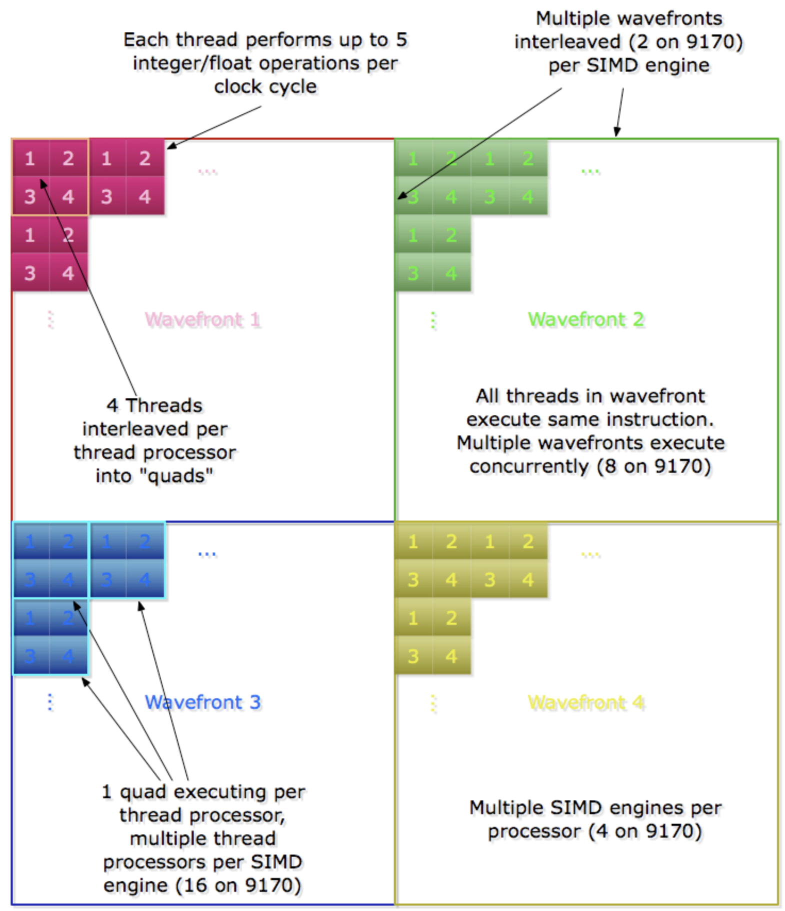
Programming AMD GPUs



Streaming Model



Mapping Streaming Model onto AMD GPUs



APIs

CAL

- Low level API for stream programming
- C frontend
  - Very verbose
- Intermediate Language (IL) kernels
  - Pseudo-assembly language
- SIMD data types

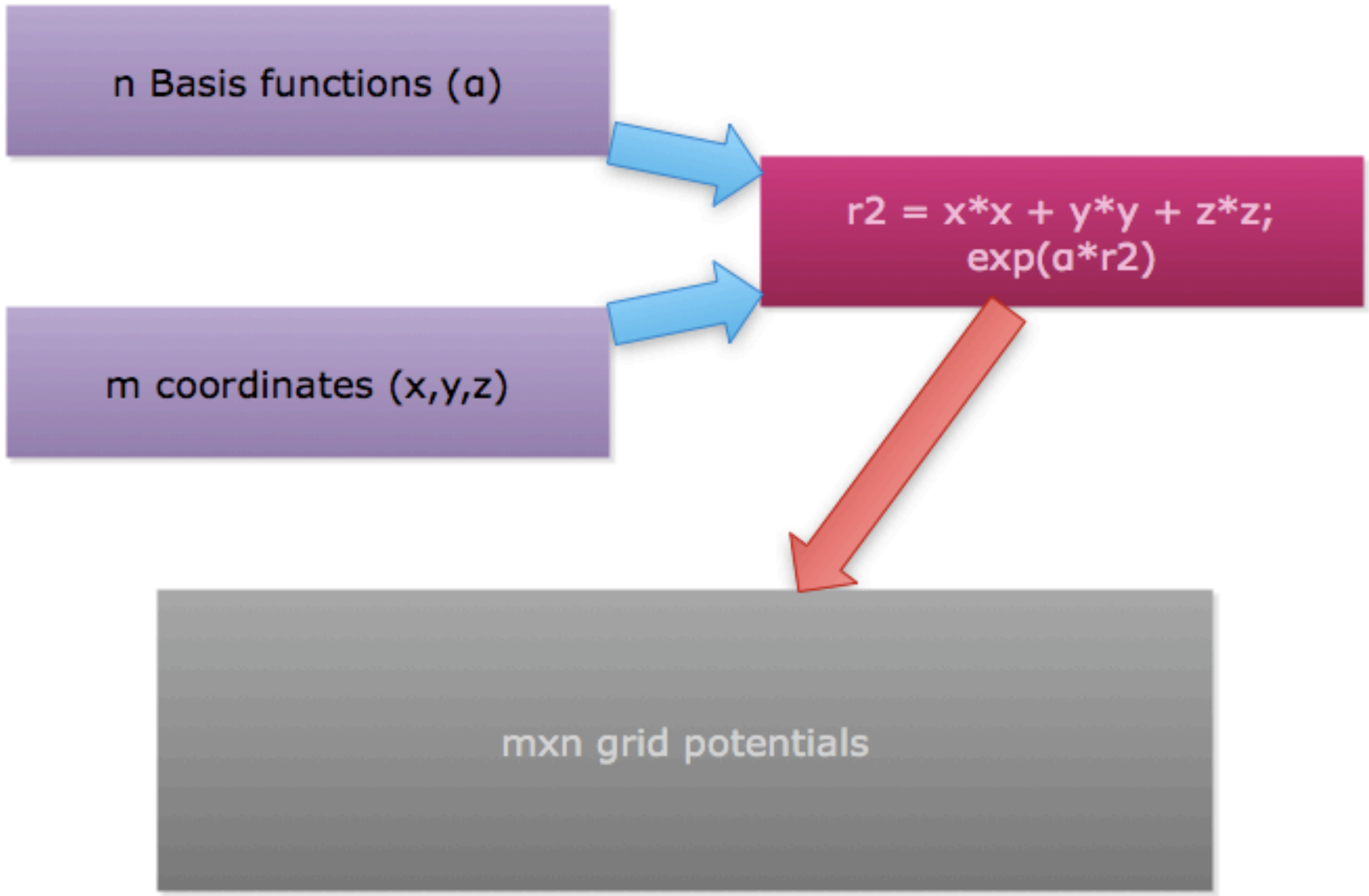
Brook+

- High level C-like language based on Brook project
- Streaming model
- Very abstract!
  - Read and write to streams

Grid Potential

- Compute Gaussian basis functions to approximate Slater functions
- Used in Hartree-Fock method to approximate ground state wave function of n-body quantum mechanical system
- exp(-ar^2), where a is spread and r^2 is distance between two orbitals

Grid Potential



Naïve C implementation

```
for(i = 0; i < npt; i++)
{
  float r = x[i] * x[i] + y[i] * y[i] + z[i] * z[i];

  for(j = 0; j < nbas; j++)
  {
    basis2[j*npt+i] = exp(alpha[j] * r);
  }
}
```

Naïve GPU implementations

Brook+

```
kernel void computeBasisFunction(float alpha[],
                                 float xCoord[],
                                 float yCoord[],
                                 float zCoord[],
                                 out float basis<>)
{
  float2 index = indexof(basis).xy;

  float x2;
  float y2;
  float z2;
  float r2;

  x2 = xCoord[index.y] * xCoord[index.y];
  y2 = yCoord[index.y] * yCoord[index.y];
  z2 = zCoord[index.y] * zCoord[index.y];

  r2 = x2 + y2 + z2;

  basis = exp(alpha[index.x] * r2);
}
```

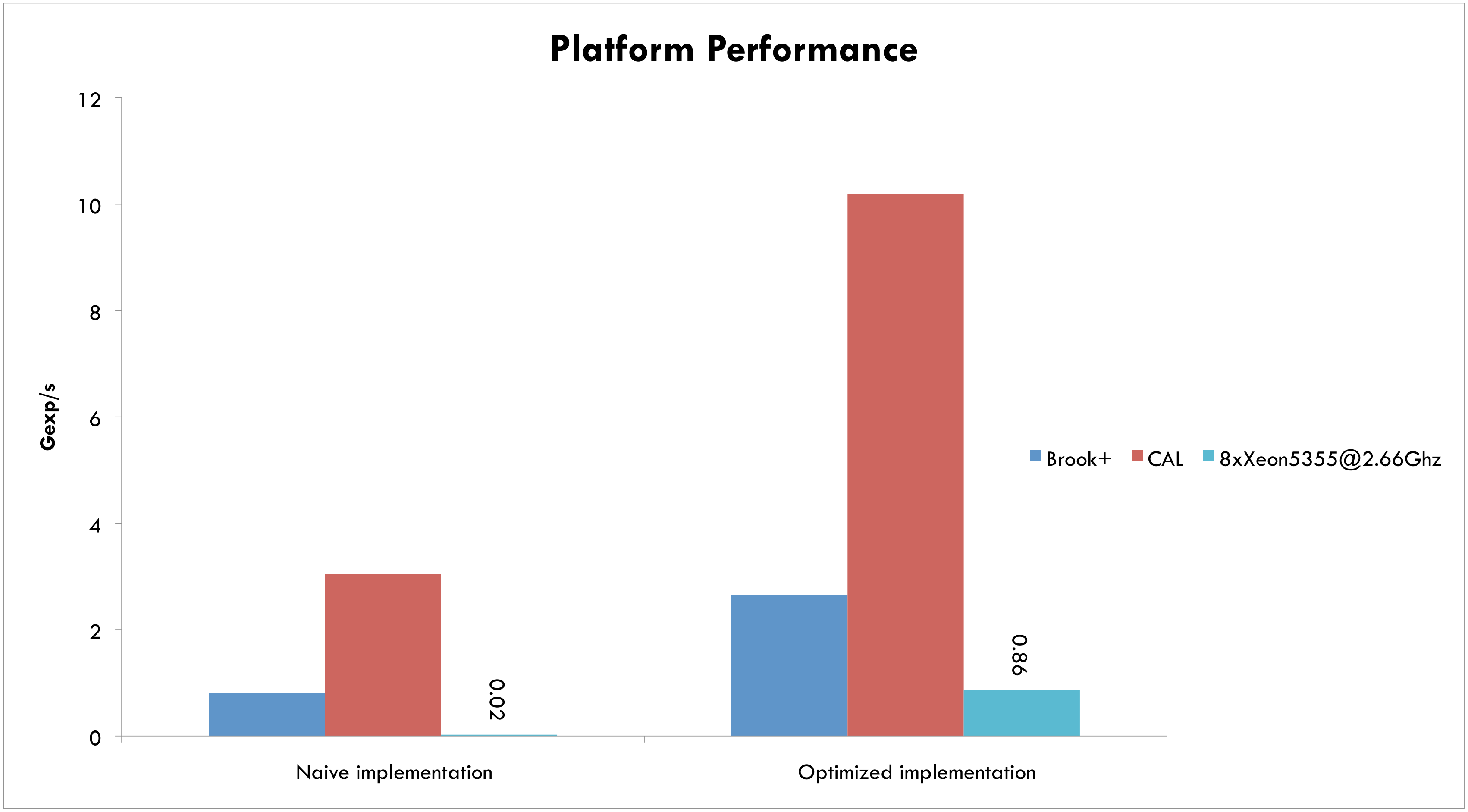
CAL

```
const char* basisFunc =
"il_ps_2_0 \n"
"dcl_output_generic o0\n"
"dcl_literal l0, 2.71828182845904523536, 2.71828182845904523536, 2.7182818284590
4523536, 2.71828182845904523536\n"
"dcl_input_position_interp(linear_noperspective) vWinCoord0.xy__\n"
"dcl_resource_id(0)_type(1d,unnorm)_fmtx(float)_fmty(float)_fmtz(float)_fmtw(flo
at)\n"
"dcl_resource_id(1)_type(1d,unnorm)_fmtx(float)_fmty(float)_fmtz(float)_fmtw(flo
at)\n"
"dcl_resource_id(2)_type(1d,unnorm)_fmtx(float)_fmty(float)_fmtz(float)_fmtw(flo
at)\n"
"dcl_resource_id(3)_type(1d,unnorm)_fmtx(float)_fmty(float)_fmtz(float)_fmtw(flo
at)\n"
"sample_resource(0)_sampler(0) r0.w, vWinCoord0.yyyy\n" //load x
"sample_resource(1)_sampler(0) r1.w, vWinCoord0.yyyy\n" //load y
"sample_resource(2)_sampler(0) r2.w, vWinCoord0.yyyy\n" //load z
"sample_resource(3)_sampler(0) r3.w, vWinCoord0.xxxx\n" //load alpha
"mul r4, r0, r0\n" //r2 = x * x
"mad r4, r1, r1, r4\n" //r2 += y * y
"mad r4, r2, r2, r4\n" //r2 += z * z
"mul r6, r4, r3\n" //alpha * r2
"pow r5, l0, r6\n" //exp(alpha * r2)
"mov o0, r5\n"
"end\n";
```

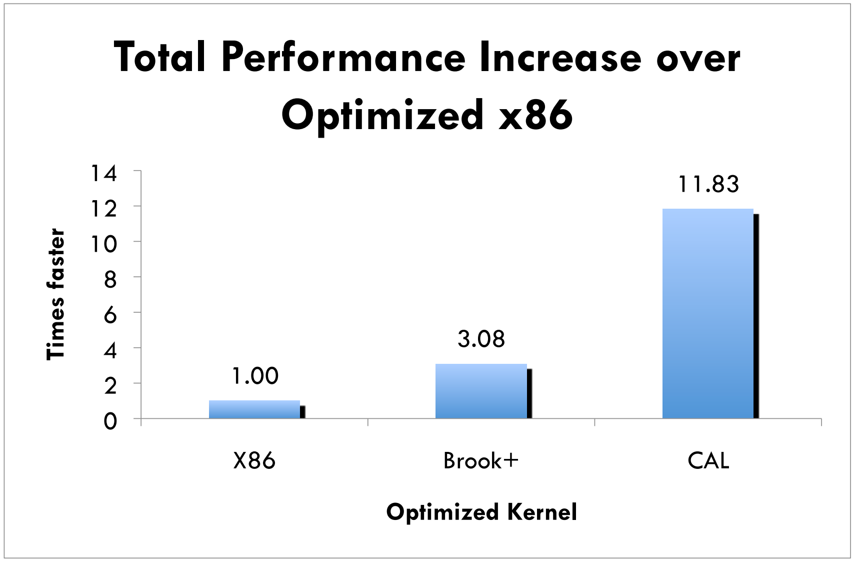
Optimizations

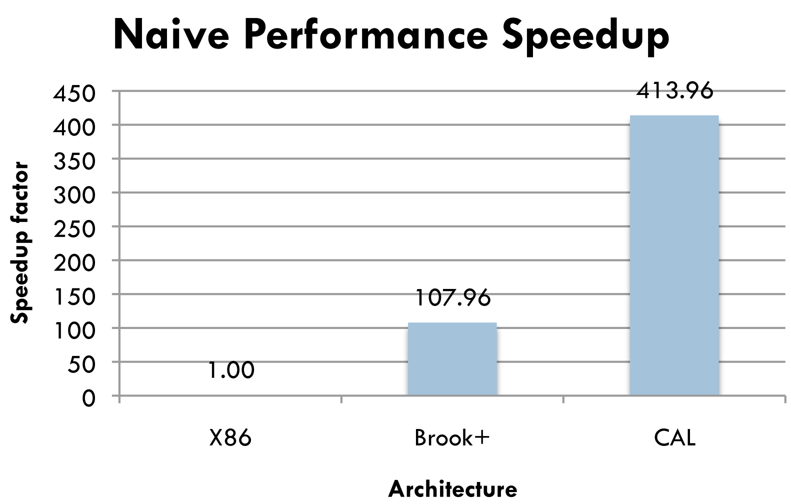
Optimization	Platform		
	x86	Brook+	CAL
Kernel unrolling	No	Yes	Yes
SIMD	Yes (Intel MKL VML)	Yes	Yes
Precompute radii	Yes	Yes	No
Cache alphas	Yes	No	No
Cache block coords	Yes	No	No

Performance in grid potentials / sec



Final Speedup





Conclusions

- Streaming is an elegant solution to SOME applications
  - Data must be completely independent
  - Data must be input or output (no inout)
- Brook+ allows for quick and dirty implementation and can give modest speedups over CPU
  - But compiler-introduced overhead hinders performance in short kernels
- CAL is difficult to program but allows more control (and hence more performance)