PARTICLE SIMULATION ON A GPU FROM PYTHON

SIAM CSE 2011
MINISYMPOSIUM ON KINETIC PLASMA MODELING

Min Ragan-Kelley (UCB AS&T)
John Verboncoeur (MSU ECE)
March 3, 2011
• Motivations for the Project
• Basics of GPUs wrt CPUs
• PIConGPU
• Python Interface
• PIC is useful for general plasma simulations
• GPUs are cheap\(^2\) and accessible performance
• PIC is (fairly) well suited to data-parallelism
• Good interfaces facilitate scientific exploration
• Python-steered GPU simulations are a good solution for scalable simulations
**WHAT IS FDTD PIC**

\[ \nabla^2 \Phi_{[i,j]} = -\frac{\rho_{[i,j]}}{\epsilon} \quad \ddot{x}(\vec{r}) = -\frac{q}{m} \nabla \Phi(\vec{r}) \]

- Particle-In-Cell (PIC) Simulations are popular in plasma physics
- E/M Fields are defined on a discrete grid
- Particles live in continuous space
- Particles do not directly interact, but rather are mediated by the grid
- PIC: Weight (p->g), Solve(g), Interpolate (g->p), Push(p)
Architectures
CPU

- massive clever Cache
- heavy OOO scheduling, branch prediction
- 2-16 cores / chip
- SMT (HyperThreading)
- 4 (SSE) - 8(AVX) wide SIMD vector ops

Intel Nehalem (Core i7)
GPU

• 10s of MPs, 100s of cores, 10,000s of threads

• no deep scheduling

• Only very basic Cache model

• 1-4GB fast device memory (~150GB/s)
Super-Cell = ThreadBlock
(1 thread = 1 cell)

Coalesced Access
Attribute Tiles (like ParticleGroup)
Size = tpb (256)
5 [7] floats per particle
5 [9] floats per gridpoint
3GB RAM per GPU
• compute E
  • send E

• Compute B (interior first, then wait for E)
  • send B

• Advance Particles (interior first, then wait for B)
  • send Particles

• Compute Source (interior first, then wait for particles)
PIConGPU scaling

![Graph showing normalized runtime against number of GPUs for strong and weak scaling scenarios.](image)

- **Strong Scaling**: (2048x2048 grid, ~28M particles)
- **Weak Scaling**: (per GPU: 2048x2048 grid, ~7.5M particles)
Now, to get the data out
THE PAST: XGRAFIX

- Object Oriented Particle-In-Cell (OOPIC)
- Flexible model, but not scalable
- Mouse-only interface during simulations
- Better tools exist now, that didn’t in 1995
• Represent Device as Python object
• Provide superset of XGrafix Controls as methods of that object
• Present existing diagnostics as numpy arrays
• Allow arbitrary derivative diagnostics
• Pure Python for input files (no custom syntax)
• Multiple Backends (OpenCL, 2D, 3D, EM, etc.)
• Diagnostic aggregation for multi-device
Use Cases
from OOPIC import *
sim = Simulation("/path/to/inputfile.inp")
defaultDiags = sim.diagnostics
sim.run() # run simulation until interrupt
sim.run(10) # run 10 steps
sim.step() # step forward once
J = sim.diagnostics["Current"]
B = sim.diagnostics["Magnetic Field"]
JcrossB = cross(J[:,6], B[:,6]) # adds JxB Diagnostic
JcrossB.interval = 10 # update every 10 timesteps
pylab.plot(sim.X, JcrossB)

def f(N):
    return N > 1e9
n = sim.diagnostics["Number"]
sim.runUntil(f, (n,)) # runs until there are 1e9 particles
• As long as you can define your analysis, you can do it.
• Second simulation depends on results of first
• sim2 could be DSMC or anybody else's code

```python
from mystuff import *

sim1 = Simulation("sim1.py")
def stop_condition(sim):
métric = check_some_stats(sim.diagnostics)
    if metric > threshold:
        return True

sim1.run_until(stop_condition, (sim1,))

analysis = perform_analysis(sim1)
del sim1
sim2 = Simulation("sim2.py")
update_with_analysis(sim2, analysis)
sim2.run()
```
• here we have a set of parameters that we want to find

• run a simulation, and get new input parameters from analysis

• start again with new parameters

• repeat until new parameters are close enough to the most recent input.
• The UI and the Analysis systems are *the same*.

• Interface Based Backend model facilitates extensions of Physics

• Python provides intuitive, extensive Sim building tools

• Ground up Parallel design will scale better than XOOPIC

• PIConGPU + IPython (ØMQ) are excellent starting points
Sources

- CS267
- ParLab Bootcamp videos and pdfs
  - http://parlab.eecs.berkeley.edu/bootcampaagenda
  - Specifically Demmel, Catanzaro, Mattson

- NVIDIA GPU Gems

- NVIDIA CUDA/OpenCL Docs

- Bussman’s PIConGPU GTC talk
THANKS

• AFOSR (funding)
• John Verboncoeur (advisor)
• Michael Bussmann (PICoNGPU)
• Fernando Perez & Brian Granger (IPython)
• Andreas Klöckner (PyCUDA/PyOpenCL)
• Pat Berumen (AS&T)
END