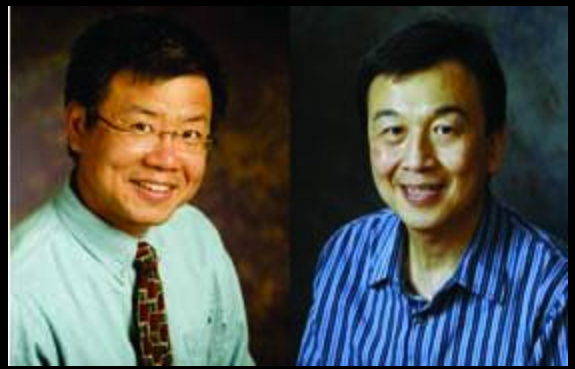
FCUDA: ENABLING EFFICIENT COMPUTATION OF CUDA KERNELS ONTO FPGAS

Alexandros Papakonstantinou, Karthik Gururaj, John A. Stratton, Deming Chen, Jason Cong, Wen-Mei W. Hwu

CS525 Presentation: Alessandro Febretti

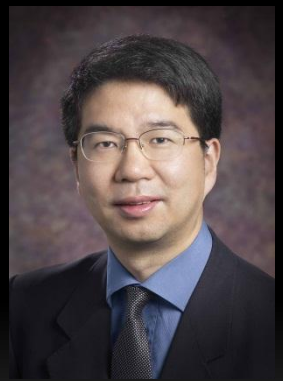
About the Paper

- Best paper award at IEEE SASP'09



Deming Chen, Wen-mei Hwu
Coordinated Science Laboratory, UIUC

Alexander Papakonstantinou, PhD Student
John Stratton, M.S.



Jason Cong
Center for Customizable Domain-Specific Computing, UCLA

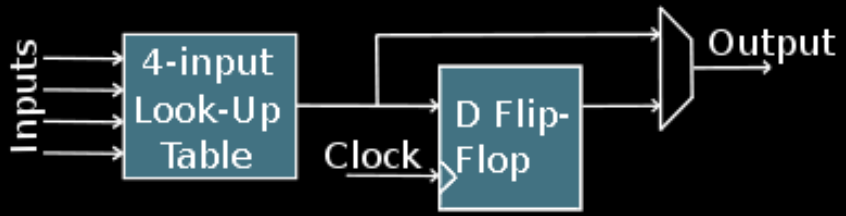
Karthik Gururaj, PhD Student

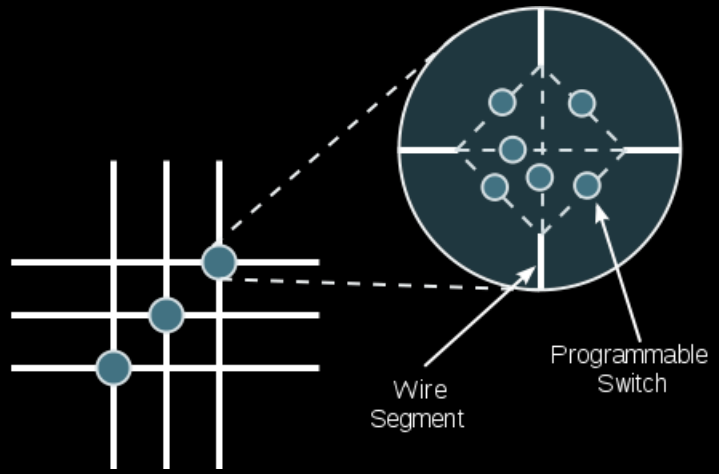
Objective

- Running CUDA kernels on **Field-programmable gate arrays**
  - Reconfigurable hardware: ICs whose logic can be modified.
  - Created as substitutes for ASICs in certain areas
  - Parallel by nature
  - High energy efficiency (~6X vs GPUs)

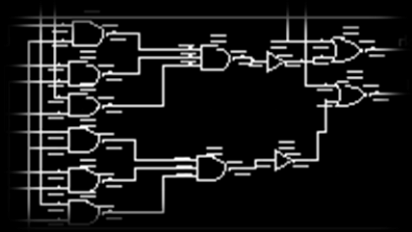
How does an FPGA work?

- Grid of Configurable Logic Blocks.
- CLBs connected by routing channels with configurable switches.
- Sometimes higher level hardware
  - DSPs, ALUs, …





How to program FPGAs

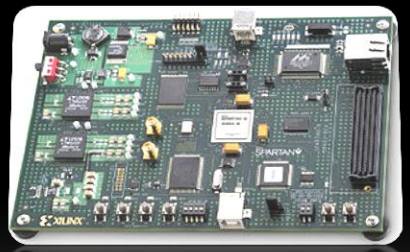


Schematic

Hardware Description Language

High Level Language

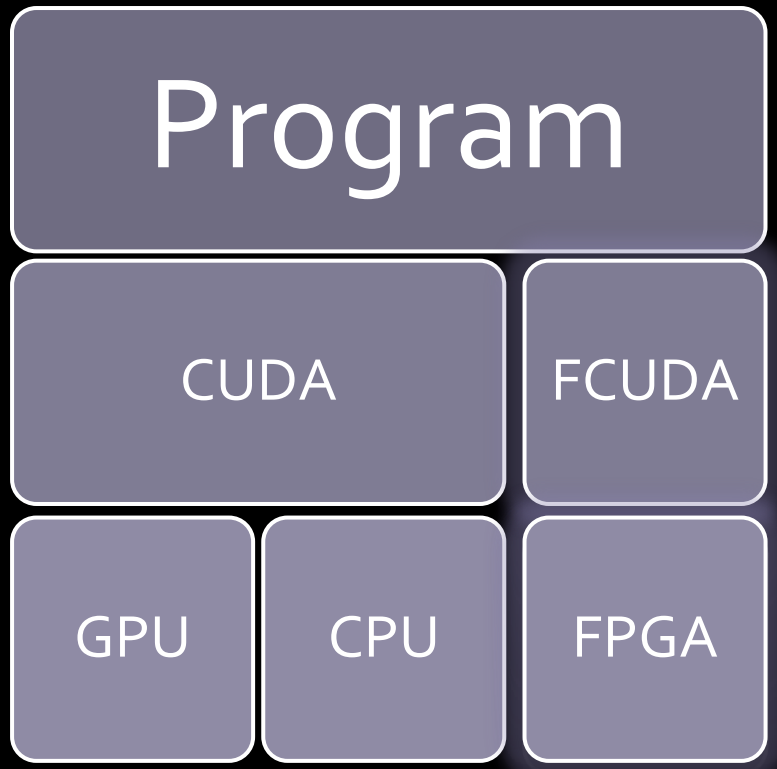
Device



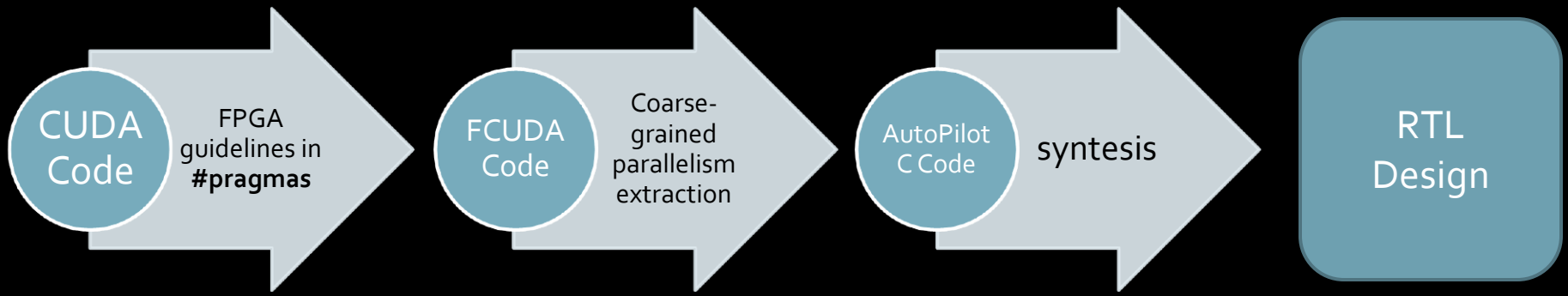
```
void CruiseControl_init(_C_CruiseControl * _C_)
{
    CruiseSpeedMgt_init(&(_C_->_C0_CruiseSpeedMgt));
    CruiseStateMgt_init(&(_C_->_C3_CruiseStateMgt));
    (_C_->_M_condact_0) = true;
    ThrottleCmd_init(&(_C_->_C4_ThrottleCmd));
    (_C_->_M_init) = true;
}

/* ========= */
/* MAIN NODE */
/* ========= */

void CruiseControl(_C_CruiseControl * _C_)
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOfLimits;
    bool _L19;
/*#code for node CruiseControl */
/* call to node not expanded DetectPedalsPressed */
    (_C_->_C1_DetectPedalsPressed._I0_Brake) = (_C_->_I
    (_C_->_C1_DetectPedalsPressed._I1_Accelerator) = (
    DetectPedalsPressed(&(_C_->_C1_DetectPedalsPressed
    BrakePressed = (_C_->_C1_DetectPedalsPressed._O0_Br
    AcceleratorPressed =
        (_C_->_C1_DetectPedalsPressed._O1_AcceleratorPr
/* call to node not expanded DetectSpeedLimits */
    (_C_->_C2_DetectSpeedLimits._I0_speed) = (_C_->_I8_
    DetectSpeedLimits(&(_C_->_C2_DetectSpeedLimits));
    SpeedOutOfLimits = (_C_->_C2_DetectSpeedLimits._O0_
/* call to node not expanded CruiseStateMgt */
```

Why FCUDA?

- Use common language for CPUs, GPUs, FPGAs
  - GPU/FPGA architectures
- Some kernels run **faster** on a FPGA
- Easy to express parallelism in CUDA

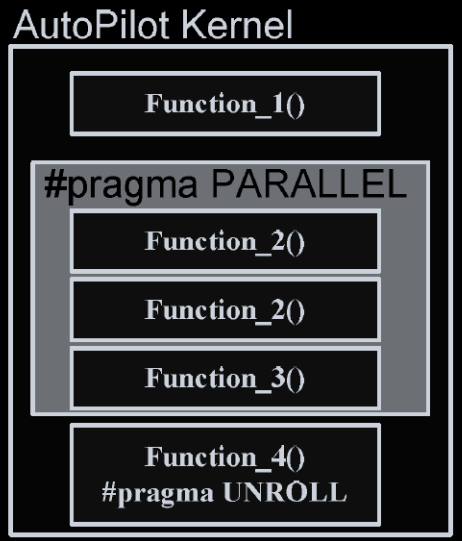


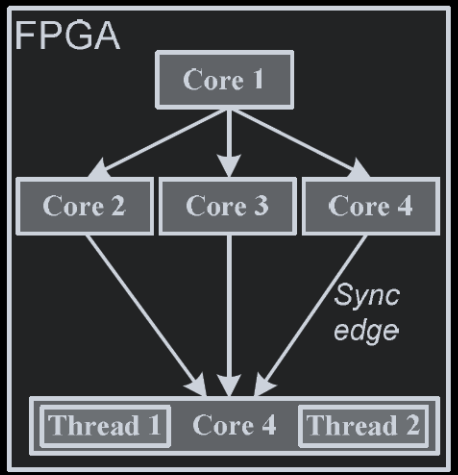
Cuda to FPGA flow



Autopilot C

- Parallelism expressed through #pragmas inside code
- Explicit sync barriers
- No thread level sync
- No shared memory

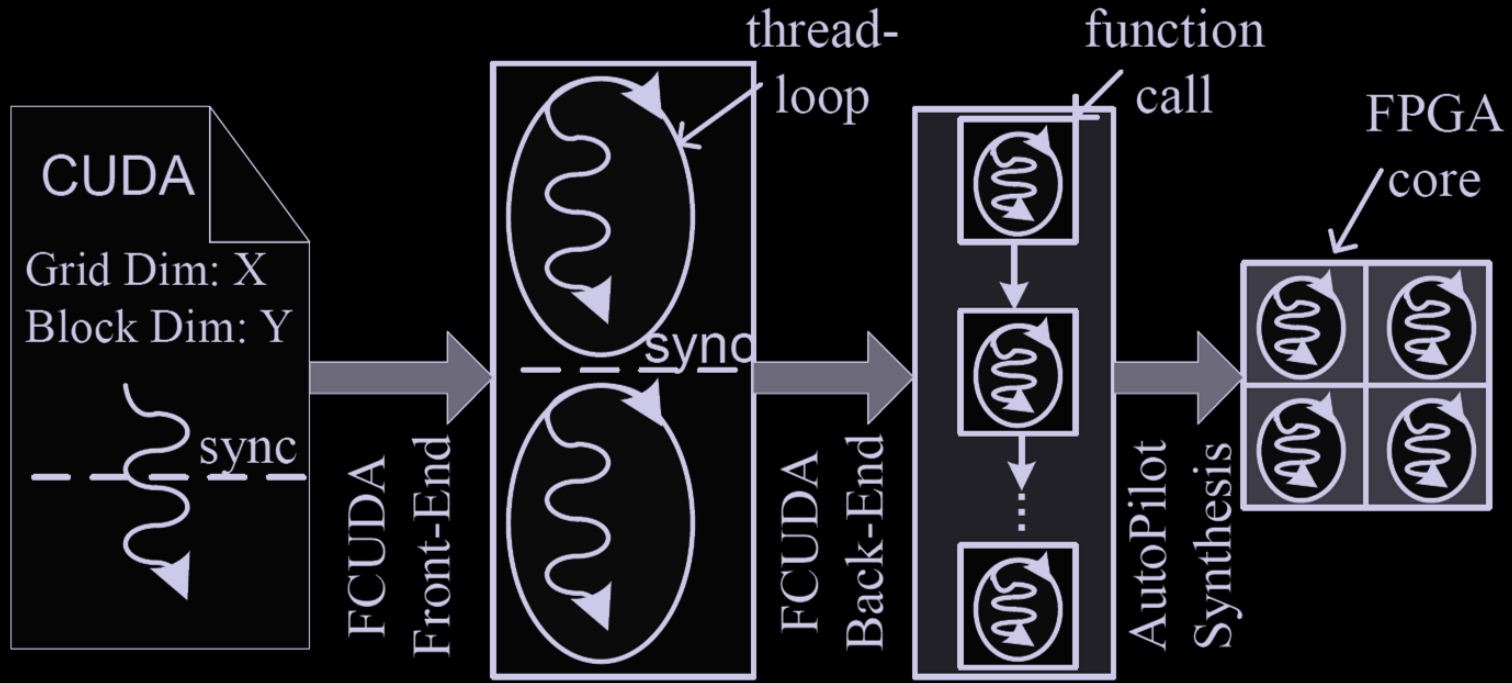




FCUDA philosophy

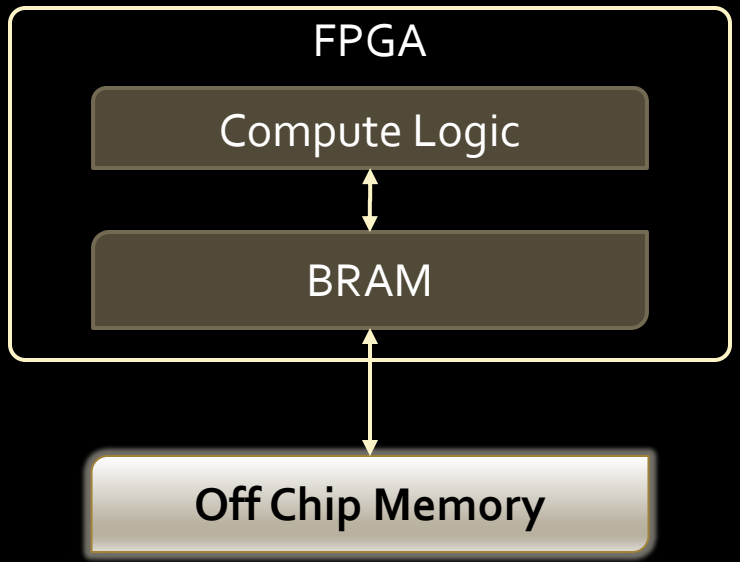
- Transform CUDA thread-blocks into parallel AutoPilot function calls.
  - Thread-block level maps well to hardware-parallel cores.
  - threads inside a block are executed **sequentially**
- Decouple computation to off-chip memory transfers
  - Avoid latency problems.

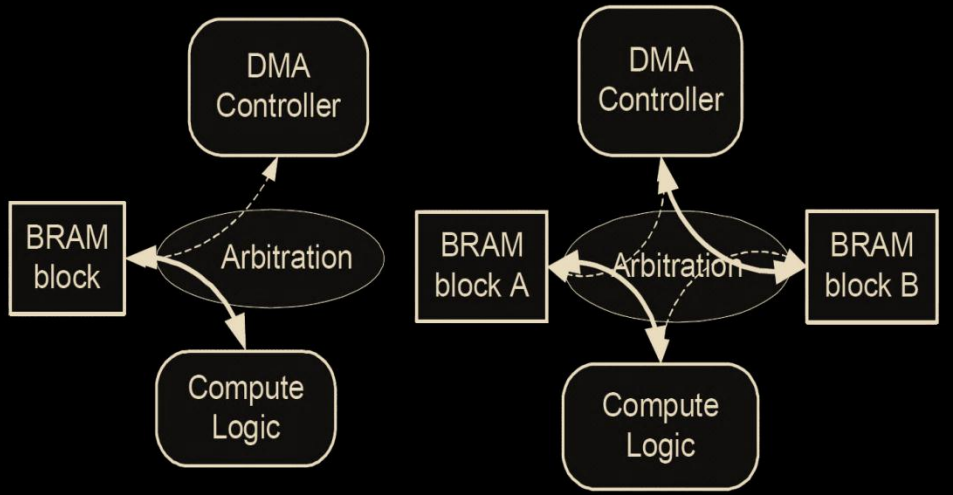
Parallelism Extraction



Memory Access

- Logic has no direct access to off-chip memory
- DMA controller transfers to-from BRAM



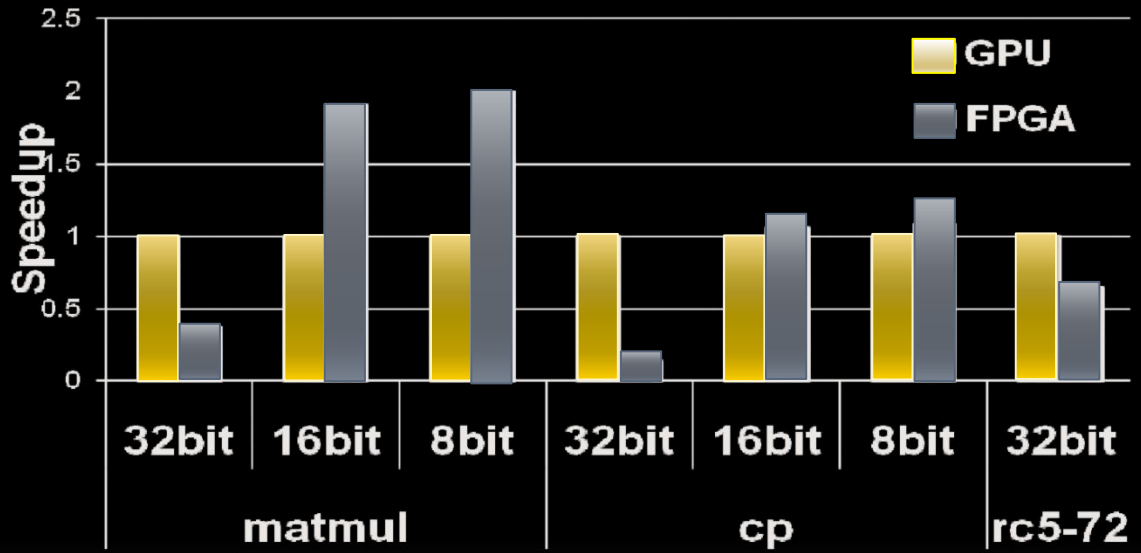


Sample annotated kernel

```
#pragma FCUDA GRID x_dim=2 y_dim=1 begin name="cp_grid"
#pragma FCUDA BLOCKS start_x=0 end_x=128 start_y=0 end_y=128
#pragma FCUDA SYNC type=simple
__global__ void cenergy(int numatoms, int gridspacing, int* energygrid)
{
      #pragma FCUDA TRANSFER cores=1 type=burst begin name="fetch"
      #pragma FCUDA DATA type=load from=atominfo start=0 end=MAXATOMS
      #pragma FCUDA TRANSFER end name="fetch"
      #pragma FCUDA COMPUTE cores=2 begin name="cp_block"
      . . .
      int energyval = 0;
      /* For each atom, compute and accumulate its contribution to
         energyval for this thread's grid point */
      for (atomid=0; atomid < numatoms; atomid++)
      {
            . . .
            energyval += atominfo[atomid].w * r_1;
      }
      #pragma FCUDA COMPUTE end name="cp_block"
      #pragma FCUDA TRANSFER cores=1 type=burst begin name="write"
      energygrid[outaddr] += energyval;
      #pragma FCUDA TRANSFER end name="write"
}
#pragma FCUDA GRID end name="cp_grid"
```

Experimental results

Kernel	Configuration	Description
Matrix Multiply (matmul)	1024x1024	Common kernel in many imaging, simulation, and scientific application
Coulombic Potential (cp)	4000 atoms, 512x512 grid	Computation of electric potential in a volume containing charged atoms.
RSA Encryption (rc5-72)	1 Billion keys	Brute force encryption key generation and matching



THANK YOU!