

LA-UR-13-24166

Approved for public release; distribution is unlimited.

Title: SKA

Author(s): Bergen, Benjamin K.

Intended for: LDRD ER Proposal

Issued: 2013-06-06



Disclaimer:

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

SKA

Ben Bergen

Applied Computer Science (CCS-7)

Justin Tripp

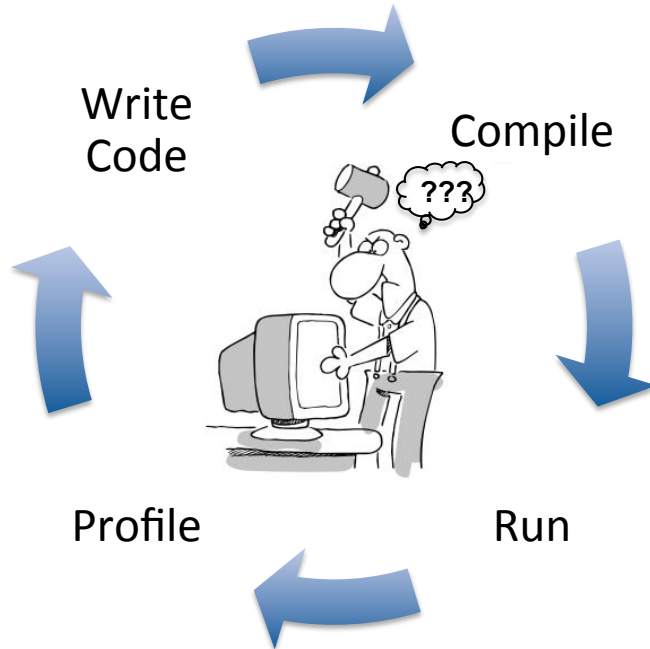
Applied Computer Science (CCS-7)

Nandakishore Santhi

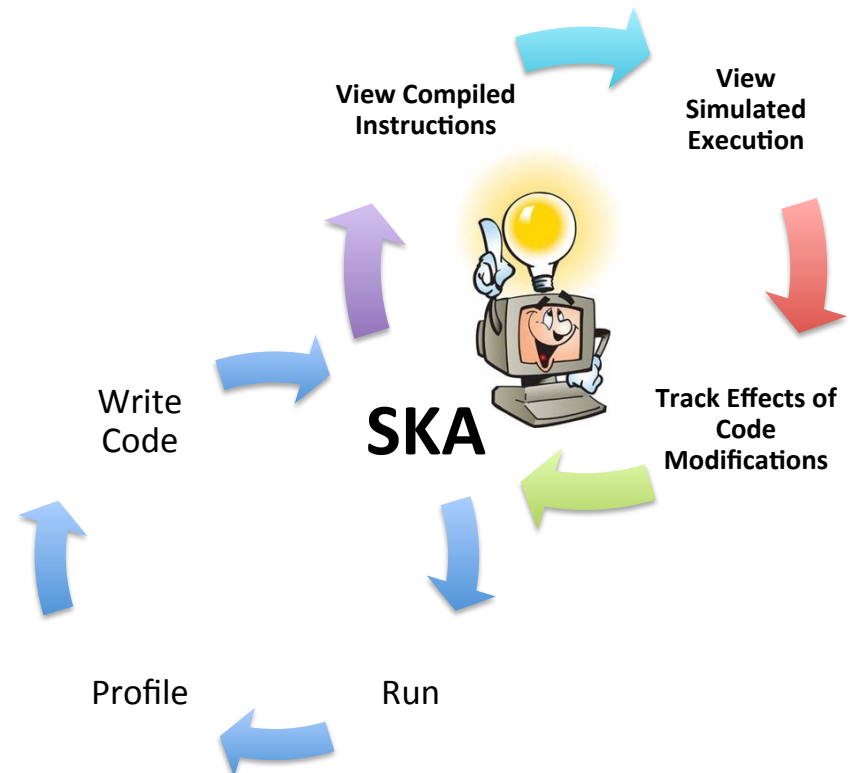
Information Sciences (CCS-3)

Why do we need SKA?

Current Development Cycle



SKA-Enhanced Development Cycle



Innovation & Timeliness

LLVM + Pipeline Simulator + Flexible Virtual Architecture

Approach finds “sweet spot” between fidelity and simulation time

- Not a full system simulator → reduced cost, more flexible
- Provides accurate simulation of important code segments

Pro Static refers to *static scheduling* (no instruction re-ordering).
a u **SKA does have moving parts!!!**

- Increase fidelity and relevance, but retain “sweet spot” strategy

Processing cores are composed of logic units...

LSU

Load/Store Unit

ALU

Arithmetic Logic Unit

ILU

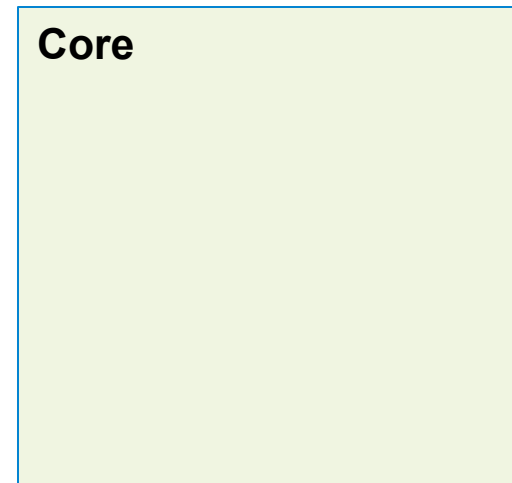
Integer Logic Unit

FPU

Floating Point Unit

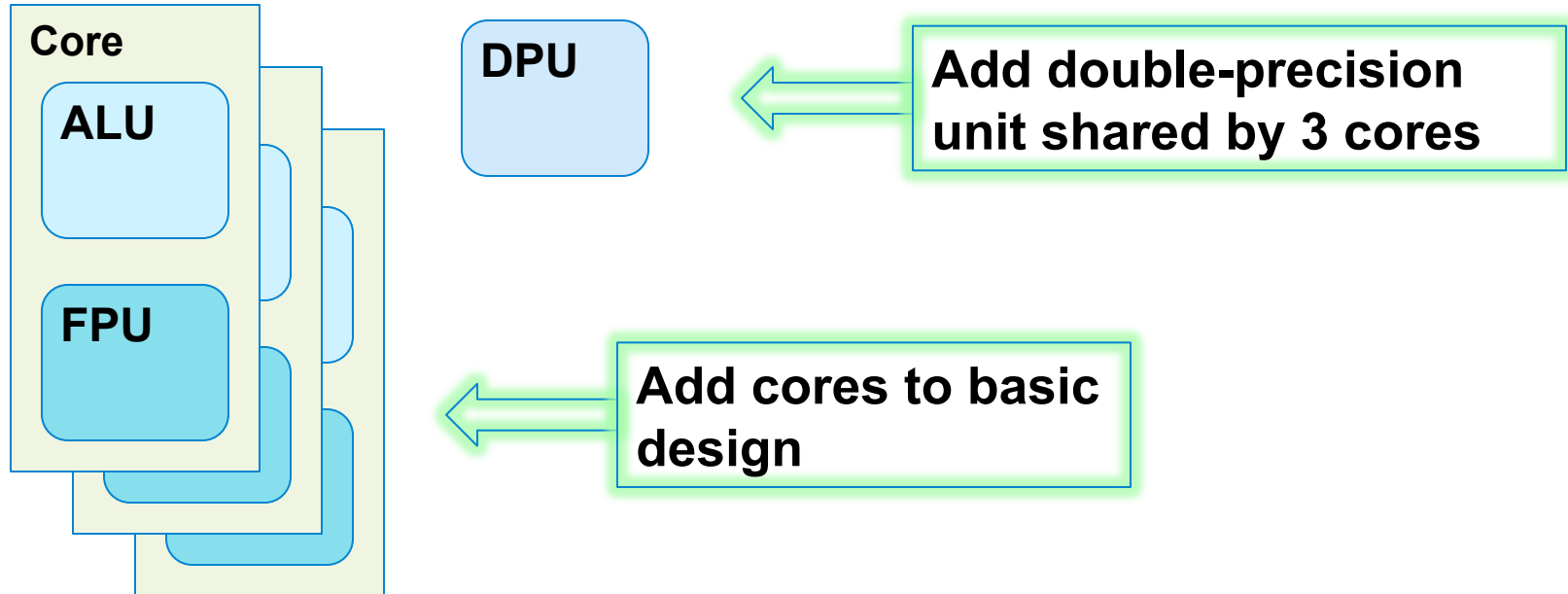
Core simulation is flexible through construction...

- ❖ **Logic units and cores have their own state**
 - ❖ Multicore/Manycore: Structural resource and cache sharing between cores will be straightforward
- ❖ **Core properties are extensible**
 - ❖ New logic unit types can be added as necessary
- ❖ **Proposed work adds support for composition of higher-level structures**



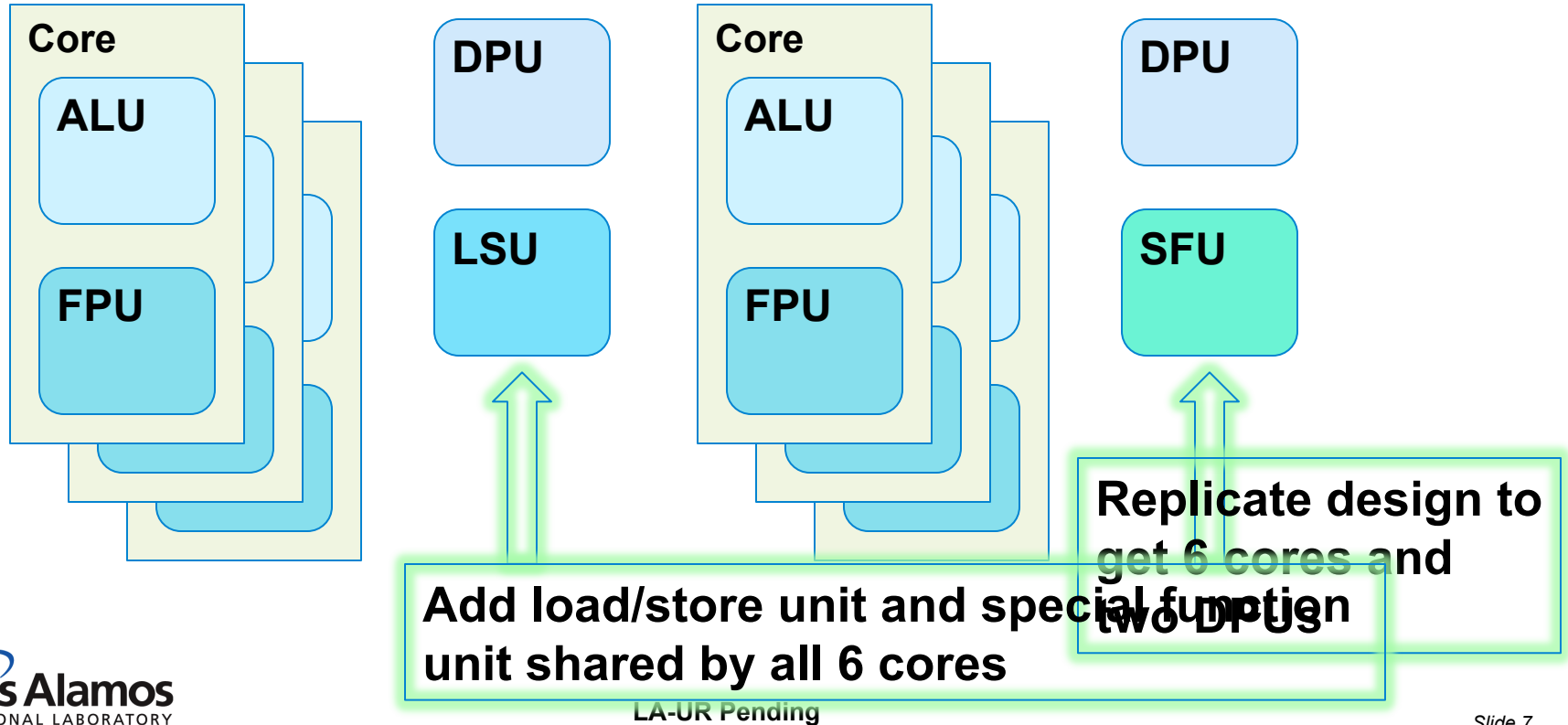
Manycore Example: NVIDIA Kepler

Architecture made up basic building blocks with shared structural resources



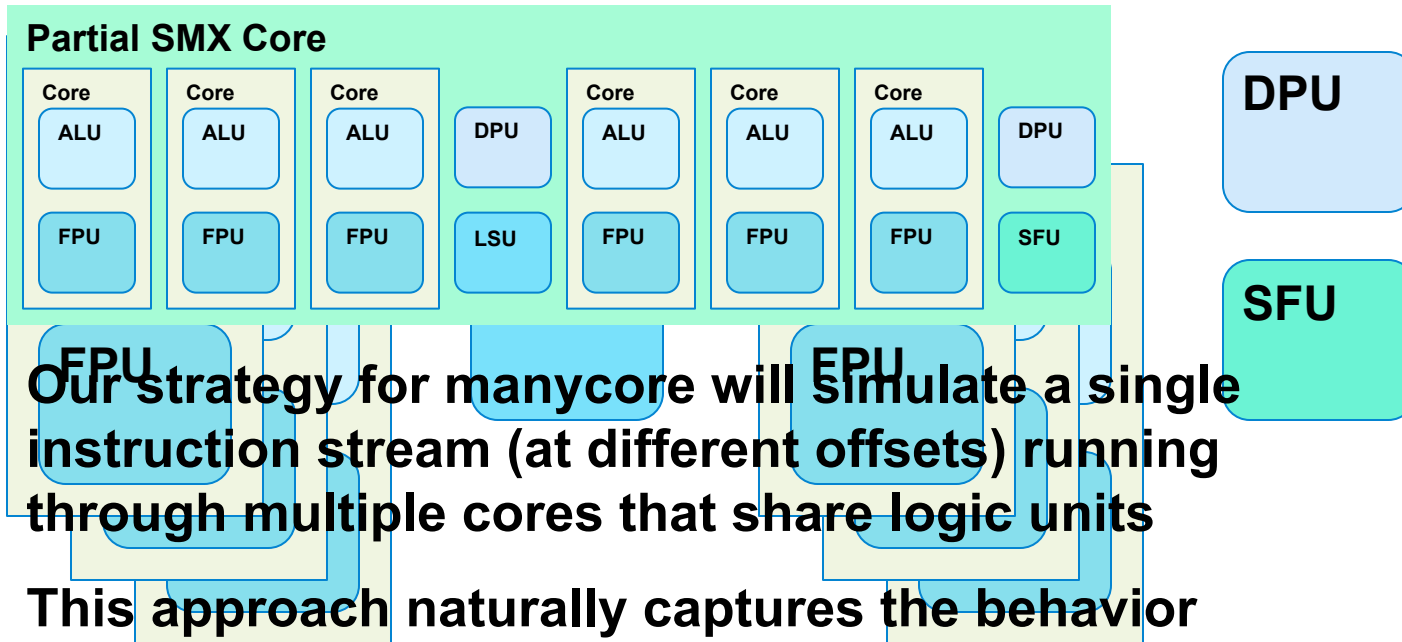
Manycore Example: NVIDIA Kepler

Architecture made up basic building blocks with shared structural resources



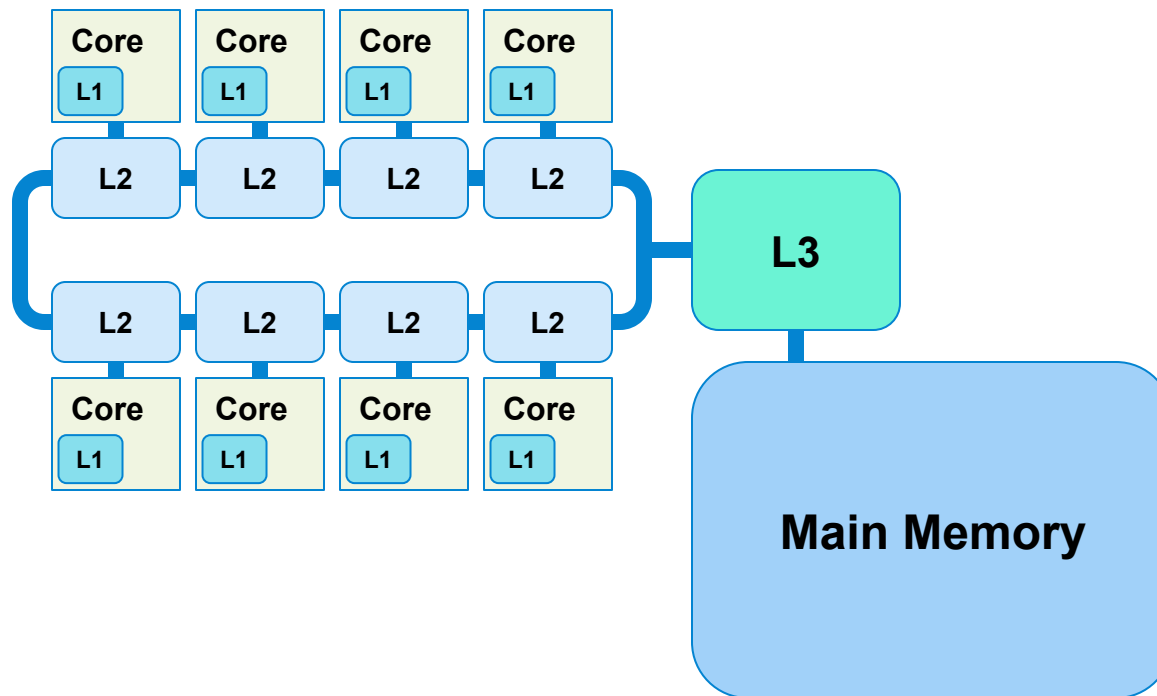
Manycore Example: NVIDIA Kepler

Architecture made up basic building blocks with shared structural resources



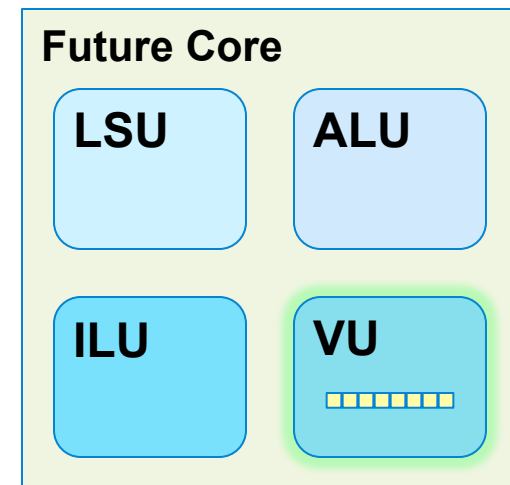
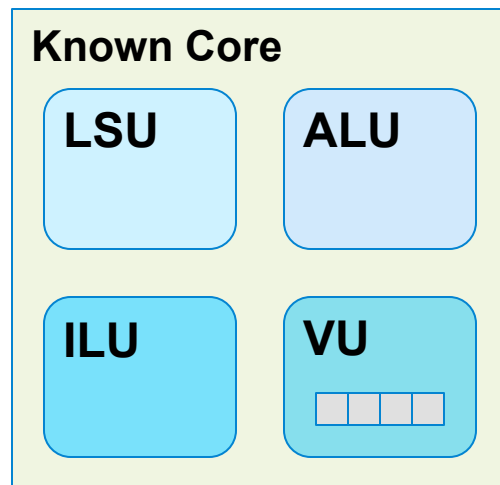
Multicore

We will employ a similar strategy for multicore using caches in addition to logic units, and with multiple instruction streams



Predictive Capability

Starting with a known, verified architectural model...



We can create a model of an architecture that is not yet available, providing simulation information before the hardware exists

Laboratory Relevance: Roxanne

- ❖ **Approximately 600,000 lines of code**
- ❖ **Complex data structures → difficult or impossible to understand data interactions or potential parallelism**

```
do nz=1,nzone_tot
  if (irdx(nz) .eq. 1) then
    nl = nlow(nz,1)
    if (level(nl) .gt. level(nz)) then
      vlx(nz,1) = vhx(nl,1)
    endif
    nh = nhgh(nz,1)
    if (level(nh) .gt. level(nz)) then
      vhx(nz,1) = vlx(nh,1)
    endif
  endif
enddo ! nz
```

Laboratory Relevance: Roxanne

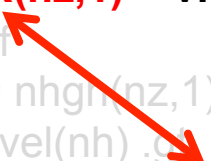
- ❖ Approximately 600,000 lines of code
- ❖ Complex data structures → difficult or impossible to understand data interactions or potential parallelism

```
do nz=1,nzone_tot
  if (irdx(nz) .eq. 1) then
    nl = nlow(nz,1)
    if (level(nl) .gt. level(nz)) then
      vlx(nz,1) = vhx(nl,1)
    endif
    nh = nhgh(nz,1)
    if (level(nh) .gt. level(nz)) then
      vhx(nz,1) = vlx(nh,1)
    endif
  endif
enddo ! nz
```

Laboratory Relevance: Roxanne

- ❖ Approximately 600,000 lines of code
- ❖ Complex data structures → difficult or impossible to understand data interactions or potential parallelism

```
do nz=1,nzone_tot
  if (irdx(nz) .eq. 1) then
    nl = nlow(nz,1)
    if (level(nl) .gt. level(nz)) then
      vlx(nz,1) = vhx(nl,1)
    endif
    nh = nhgh(nz,1)
    if (level(nh) .gt. level(nz)) then
      vhx(nz,1) = vlx(nh,1)
    endif
  endif
enddo ! nz
```



END

Execution Example

Example Simulation

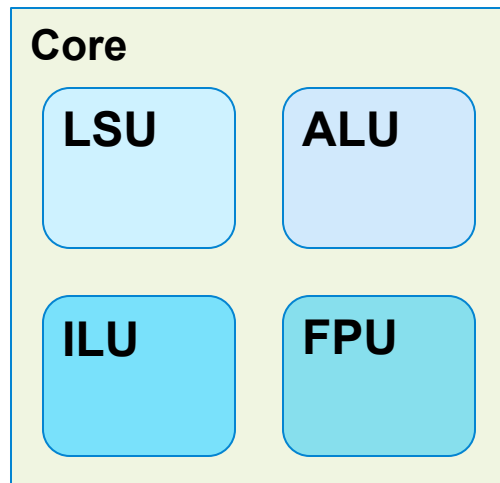
Cycle -

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6

Inst 2: %6 = **fmul** float %3, %5

Inst 1: %5 **load** float* %4, align 4



FPU Pipeline

Example Simulation

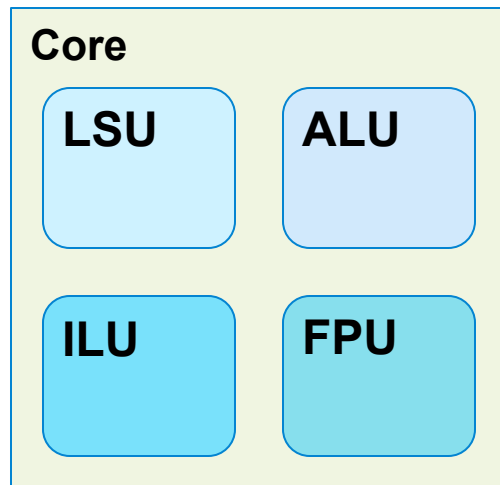
Cycle -

Inst 4: store float %7, float* %4, align 4

Inst 3: %7 ← fadd float %4, %6

Inst 2: %6 ← fmul float %3, %5

Inst 1: %5 ← load float* %4, align 4



FPU Pipeline

Example Simulation

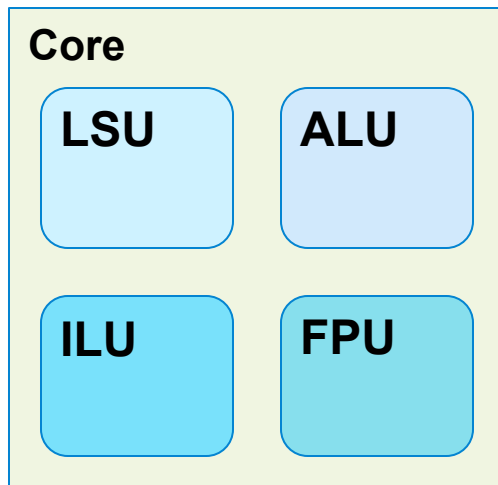
Cycle -

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6

Inst 2: %6 = **fmul** float %3, %5

Inst 1: %5 **load** float* %4, align 4



FPU Pipeline

Example Simulation

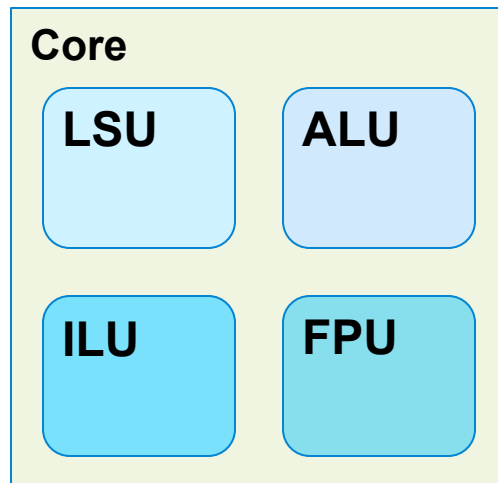
Cycle -

Inst 4: store float %7, float* %4, align 4

Inst 3: %7 = fadd float %1, %6

Inst 2: %6 = fmul float %3, %5

Inst 1: %5 load float* %4, align 4



FPU Pipeline

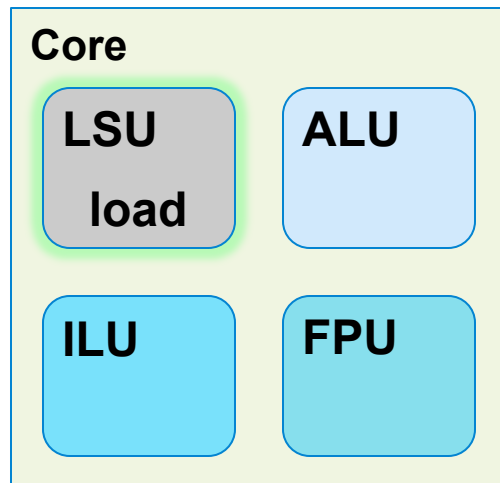
Example Simulation

Cycle 0

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6

Inst 2: %6 = **fmul** float %3, %5



FPU Pipeline

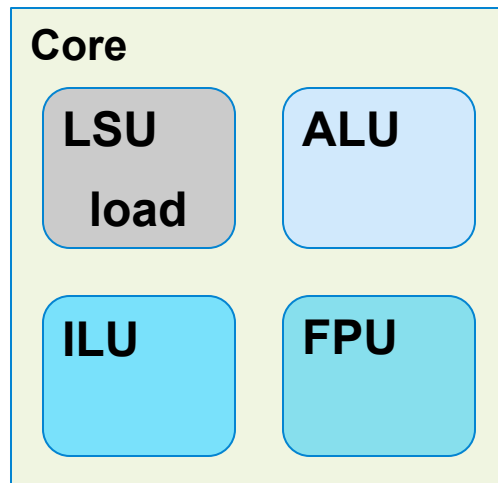
Example Simulation

Cycle 0

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6

Inst 2: %6 = **fmul** float %3, %5



FPU Pipeline

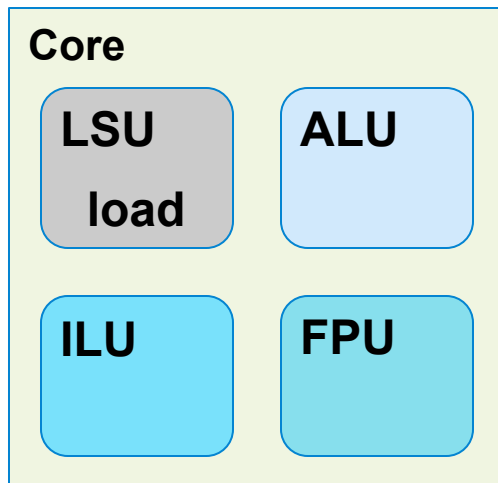
Example Simulation

Cycle 0

Inst 4: store float %7, float* %4, align 4

Inst 3: %7 = fadd float %1, %6

Inst 2: %6 = fmul float %3, %5



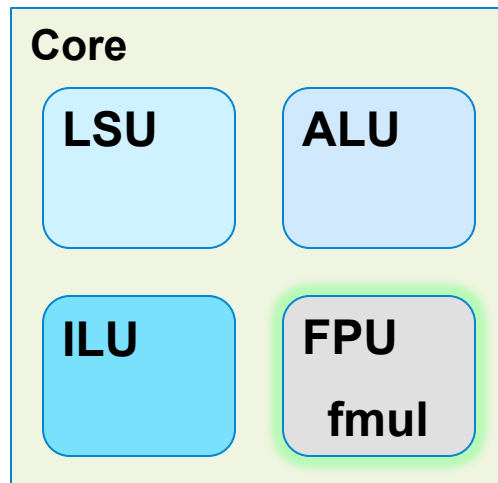
FPU Pipeline

Example Simulation

Cycle 1

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6



FPU Pipeline

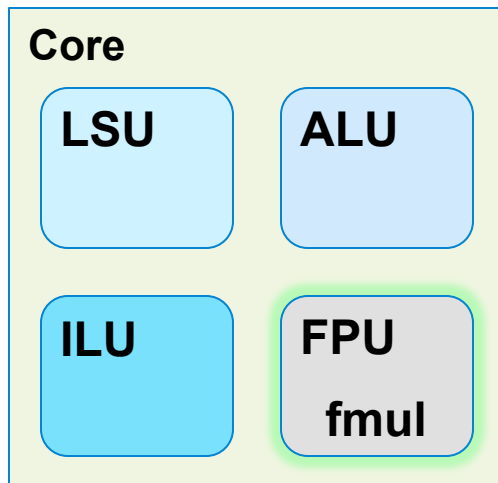
Inst 2	1						

Example Simulation

Cycle 2

Inst 4: store float %7, float* %4, align 4

Inst 3: %7 = fadd float %1, %6



FPU Pipeline

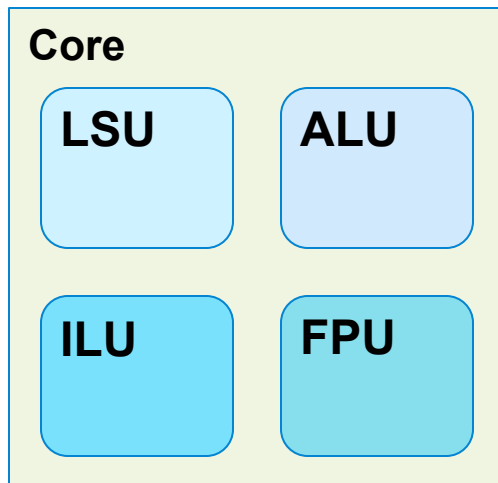
Inst 2	1	2					

Example Simulation

Cycle 3

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6



FPU Pipeline

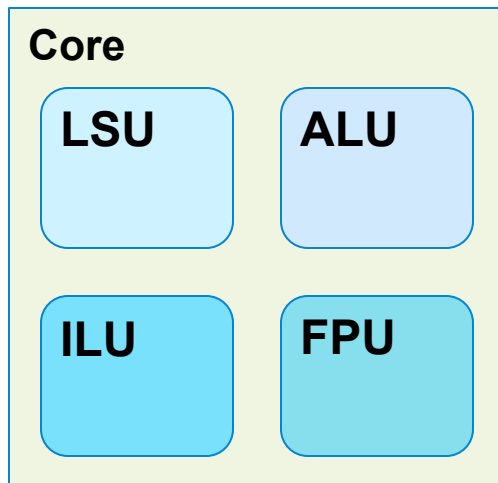
Inst 2	1	2	3				
			-				

Example Simulation

Cycle 4

Inst 4: **store** float %7, float* %4, align 4

Inst 3: %7 = **fadd** float %1, %6



FPU Pipeline

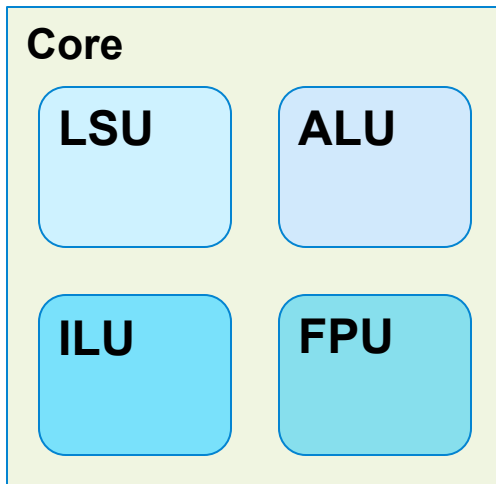
Inst 2	1	2	3	4			
			-	-			

Example Simulation

Cycle 4

Inst 4: store float %7, float* %4, align 4

Inst 3: %7 = fadd float %1, %6



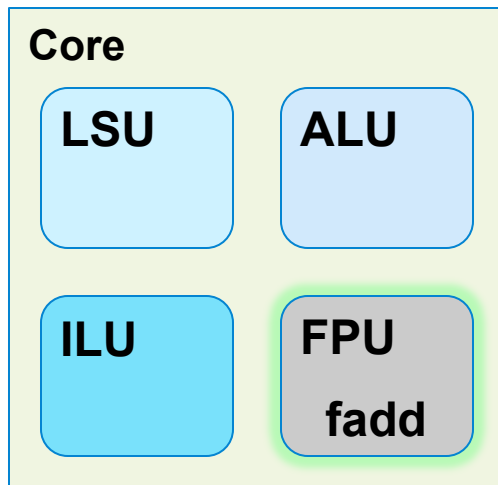
FPU Pipeline

Inst 2	1	2	3	4			
			-	-			

Example Simulation

Cycle 5

Inst 4: **store** float %7, float* %4, align 4



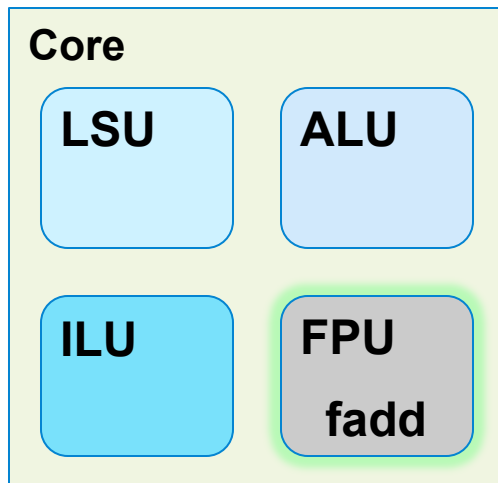
FPU Pipeline

Inst 2	1	2	3	4			
			-	-	5		

Example Simulation

Cycle 6

Inst 4: store float %7, float* %4, align 4



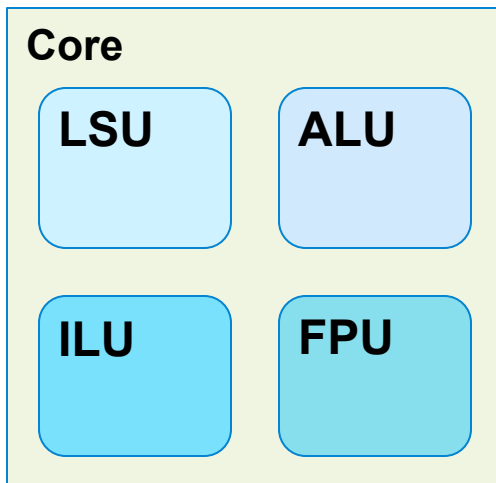
FPU Pipeline

Inst 2	1	2	3	4			
			-	-	5	6	

Example Simulation

Cycle 7

Inst 4: store float %7, float* %4, align 4



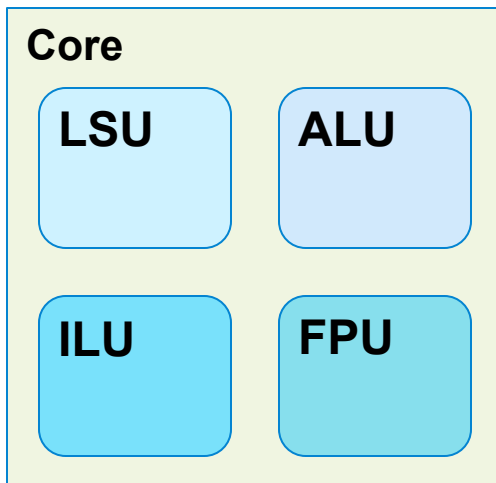
FPU Pipeline

Inst 2	1	2	3	4			
			-	-	5	6	7

Example Simulation

Cycle 7

Inst 4: store float %7, float* %4, align 4

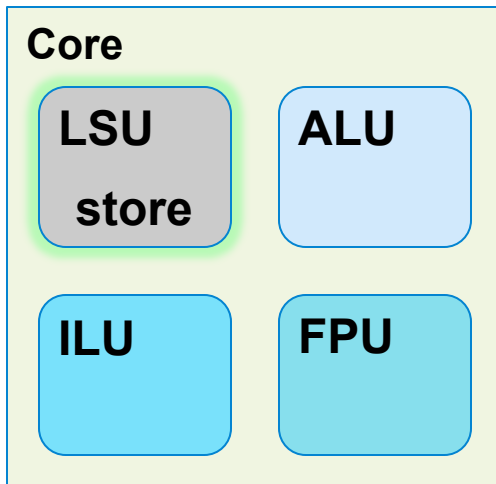


FPU Pipeline

Inst 2	1	2	3	4			
			-	-	5	6	7

Example Simulation

Cycle 8

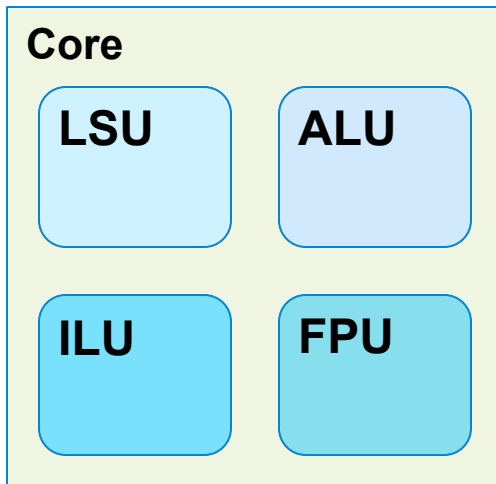


FPU Pipeline

Inst 2	1	2	3	4			
			-	-	5	6	7

Example Simulation

END



FPU Pipeline

Inst 2	1	2	3	4			
			-	-	5	6	7

Accessibility: User Interface

High-Level Interface

Function Annotation

- + MUSCLHancockHLL<MinMod>::init
- + MUSCLHancockHLL<MinMod>::dt
- + MUSCLHancockHLL<MinMod>::advance
- + MUSCLHancockHLL<MinMod>::enforceBoundaries
- + MUSCLHancockHLL<MinMod>::updateGhostBuffers
- + MUSCLHancockHLL<MinMod>::updateWaveSpeeds
- + DampedJacobi::smooth
- + DampedJacobi::enforceBoundaries
- + DampedJacobi::updateGhostBuffers
- + FullMultigrid<Smoother, DirectSolver>::solve

High-Level Interface

Function Annotation

```
+ MUSCLHancockHLL<MinMod>::init
+ MUSCLHancockHLL<MinMod>::dt
- MUSCLHancockHLL<MinMod>::advance
  Score: 82%
  Cycles: 407
  Stalled Cycles: 131
  Flops: 23
  Parallelism: 6.625
  Strahler: 2
+ MUSCLHancockHLL<MinMod>::enforceBoundaries
```

Mid-Level Interface

Kernel Annotation

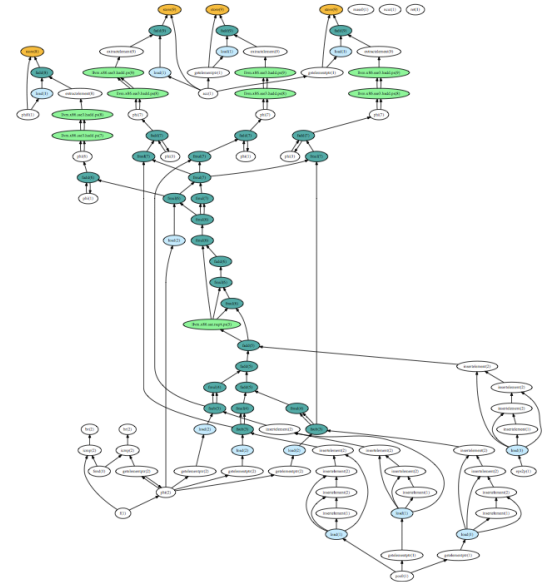
```
16  for(i=0; i<1024; f+=4, ++i) {  
17      x = ppos0 - f[1];  
18      y = ppos1 - f[2];  
19      z = ppos2 - f[3];  
20      r2 = x*x;  
21      r2 += y*y;  
22      r2 += z*z;  
23      rinv = 1.0/sqrt(r2);  
24      t = rinv*rinv;  
25      rinv *= f[0];  
26      phi += rinv;  
27      t *= rinv;  
28      a0 += x*t;  
29      a1 += y*t;  
30      a2 += z*t;  
31  } // for
```

Line 22:

Line 23:

Data Hazard Stall (56 cycles)

Structural Hazard Stall (56 cycles)



- ❖ Shows issue cycle, logic unit, multiple issue, data and structural hazards, and IR instruction stream

- ❖ Shows instruction dependency tree with colorized nodes to identify important operations
- ❖ Nodes labeled with metric information, e.g. *Strahler* number